НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Кваліфікаційна наукова праця
на правах рукопису

ГУСЄВ ЄВГЕН ІГОРОВИЧ

УДК 004.04 (043.3)

ДИСЕРТАЦІЯ
СПОСОБИ ОРГАНІЗАЦІЇ СУМІСНОГО ДОСТУПУ ДО РОЗПОДІЛЕНИХ СТОРІНОК ПАМ'ЯТІ В СИСТЕМАХ ХМАРНИХ ОБЧИСЛЕНЬ

Спеціальність 05.13.05 – Комп’ютерні системи та компоненти
Галузь знань – Інформаційні технології

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Є.І. Гусєв

Науковий керівник Кулаков Юрій Олексійович,
доктор технічних наук, професор

Київ –2017

АНОТАЦІЯ

Гусєв Є.І. Способи організації спільного доступу до розподіленим сторінок пам'яті в системах хмарних обчислень. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.05 - Комп'ютерні системи та компоненти. - Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського», Київ 2017.

Дисертація присвячена проблемі оптимізації роботи з спільними ресурсами в глобальних системах хмарних обчислень. В умовах зростаючих вимог до обсягів накопичення, перетворення і зберігання інформації, виникає тенденція щодо перенесення баз даних в хмарне середовище. Однак особливості обробки спільного ресурсу в сучасних СКБД висувають підвищені вимоги по часу відгуку до каналів передачі даних між вузлами СКБД. У праці розглянуто *системи, що ґрунтуються на реплікації змін*, системи, які використовують *shared nothing*, s*hared disk* або *shared everything* архітектуру. В результаті порівняння ефективності масштабування систем обґрунтовано доцільність подальшого удосконалення способів доступу, які базується на *shared everything* архітектурі. В якості прототипної моделі для дослідження *shared everything* архітектури була обрана система Oracle Real Application Clusters (Oracle RAC), оскільки вона має низку істотних переваг перед іншими відомими системами і має широкий спектр впроваджень, що забезпечує поширеність системи і впливає на якість її підтримки.

Серед існуючих проблем архітектури розглянуто перевантаження, тобто ситуацію, коли тривалість обробки заявки перевищує довжину черги, що з’являється протягом обробки. Обґрунтовано, що перевантаження є найбільш стримуючим фактором використання глобальних мереж в

системах хмарних обчислень. Визначено, що дослідження перевантажень у системах, що використовують *shared everything* архітектуру, а також розробка способів, які унеможливлюють або значно знижують ймовірність появи перевантажень, є найбільш актуальним питанням в контексті задачі масштабування СКБД у системах хмарних обчислень.

Для оцінки ефективності традиційних способів доступу до розподілених сторінок пам'яті та існуючих способів розв'язання конфліктів за ресурс автором запропонована трикомпонентна математична модель. Перший компонент – модель затримок, що описує сценарії обробки сторінок пам'яті, необхідна для визначення тривалості доступу до сторінок, з урахуванням забезпечення когерентності. Другий компонент – модель конфліктів – відображає реалізований в прототипній системі спосіб блокування спільного ресурсу в контексті взаємозв'язків цих ресурсів та транзакцій. Та третій – модель трафіків – головною метою якої є виділення значущих з точки зору дослідження розподілених СКБД класів ресурсів, формалізація взаємозалежностей ресурсів і транзакцій та визначення вхідних параметрів для моделі конфліктів і моделі затримок. Запропонована математична модель допомагає визначити найбільш перспективні, з точки зору оптимізації, категорії затримок, в контексті дослідження перевантажень, пов'язаних з конфліктами за спільні ресурси або розподілені сторінки пам'яті, що відкриває можливості для створення нових способів доступу до спільних ресурсів та розподілених сторінок пам'яті.

В результаті аналізу за допомогою математичної моделі недоліків існуючого способу обробки спільного ресурсу, у праці запропоновано спосіб доступу до спільних ресурсів, який порівняно з традиційним, дозволяє виключити перевантаження, що пов'язані з блокуваннями, знизити вимоги до каналів передачі даних як за часом відгуку так і по пропускній здатності при розгортанні хмарного середовища. Застосування

пропонованих новацій дозволяє розглядати в якості транспорту в хмарних СКБД не тільки високопродуктивні комутатори, розташовані всередині одного ЦОДа, а й глобальну мережу, що об'єднує ресурси декількох ЦОДів, в тому числі розподілених географічно.

Пропонований спосіб доступу успішно доповнює існуючі способи, які використовується в сучасних хмарних СКБД на базі *shared everything* архітектури, що передбачає включення в себе традиційних алгоритмів доступу, які успішно зарекомендували себе в реалізації *shared everything* архітектури (на прикладі Oracle RAC). Це дозволяє динамічно обирати найбільш адекватний, для трафіка, що оброблюється, алгоритм як з набору традиційних процедур доступу, так і інноваційних, запропонованих в роботі.

Основний акцент пропонованих новацій присвячений проблемі виникнення перевантажень при доступі до спільних ресурсів. У роботі обґрунтовується необхідність скорочення часу відгуку для зниження ймовірності перевантаження. Для вирішення проблеми пропонуються 2 підходи реалізовані в трьох методах. Перший підхід реалізований в методі двофазного виконання транзакції і методі конвеєризації фаз. Пропонується використовувати версії сторінок, які були закешовані в процесі попередньої обробки, при визначенні списку необхідних спільних ресурсів. Використання цих сторінок даних – оптимістичний підхід, оскільки сторінки інвалідовані, і, можливо, саме через дані, які запитуються. Але оптимістичність дозволяє асинхронно розіслати запити на спільні ресурси не організовуючи ланцюжок послідовних звернень, що збільшують час відгуку. Це доцільно також і тому, що оцінка часу, що витрачається на визначення взаємозалежності транзакцій і ресурсів не набагато менше порівняно з часом виконання транзакції. Таким чином, перший раз (перша фаза) в транзакції використовуються неактуальні сторінки. Завершальним етапом методу (другою фазою) є повторне виконання послідовності

операцій транзакції з уже отриманими актуальними сторінками, а потім перевірка і виправлення помилок що виникли через оптимістичний підхід першої фази для задоволення вимог ACID. Різниця методів, які реалізують підхід в тому, що конвеєризація розглянутих фаз показує кращу ефективність таких транзакцій, що включають слабо закешовані ресурси. «Сплатою» за виграш у часі виходить більш ніж дворазове збільшення обсягу операцій на кожному вузлі, що вимагає визначення області доцільності застосування методів.

Другий підхід - метод призначення обробника ресурсу дозволяє зменшити черги до сторінок, які виникають при високих інтенсивностях звернення до цих сторінок. Ця проблема при традиційному (реалізованому в Oracle RAC) способі доступу *shared everything*, висуває високі вимоги до часу відгуку між вузлами, і, в контексті перенесення в хмарне середовище, є ключовою. Суть методу - зафіксувати для кожної «гарячої» сторінки вузол обробки, мінімізувавши таким чином кількість мережевих пересилань при послідовній обробці ресурсів в черзі. Негативним ефектом методу є збільшення обсягу міжвузлового трафіку.

У четвертому розділі моделюються 3 види трафіків - перші 2 запропоновані автором для більш повного висвітлення особливостей пропонованих новацій, а 3й є синтетичним тестом (TPC BENCHMARK Standard Specification Revision 5.11) широко використовуваним для визначення продуктивності програмно-апаратних комплексів в IT індустрії. Для розглянутих трафіків проведена експериментальна перевірка запропонованої моделі та оцінка очікуваного ефекту від застосування розроблених процедур доступу та розв'язання конфліктів, що ґрунтуються на запропонованих методах. Експериментально модель порівнюється з реально функціонуючим Oracle RAC, що дозволяє обгрунтувати її адекватність. Внесення змін у математичну модель для оцінки тривалостей обробки запропонованих процедур дозволяє порівняти ефективність

традиційного та модернізованого у цій праці способу доступу до ресурсів та сторінок. Протягом дослідження параметри моделі, що відповідають фізичним параметрам системи, фіксуються таким чином, щоб емулювати глобальну мережу в якості транспорту. У якості головної вхідної метрики для всіх трафіків розглядається середня інтенсивність звернень. В залежності від розподілу класів транзакцій всередині трафіків та взаємозалежностей класів ресурсів, що складають класи транзакцій, визначаються інтенсивності перевантаження, тобто граничні інтенсивності в яких система ще залишається у робочому стані для кожного трафіку.

У всіх випадках оцінка можливого ефекту була від 35% до 913%. Такий результат створює передумови для використання глобальних мереж в якості транспорту, але слід зазначити, що характеристики трафіку можуть значною мірою вплинути на ймовірність перевантаження. Таким чином, запропонований спосіб може посісти чинне місце в арсеналі засобів масштабування хмарних систем.

Ключові слова: Хмарні обчислення, СКБД, Oracle RAC, перевантаження, блокування, черга, сторінка пам'яті, shared everything архітектура.

Список публікацій здобувача

1. Гусев Е.И. Моделирование способов организации доступа к распределённым страницам памяти в системах облачных вычислений основанных на shared everything архитектуре / Е.И. Гусев // Вісник НТУУ «КПІ» .Сер. Інформатика, управління та обчислювальна техніка. – 2016. – Вип. 64. – С.133-137.

Реферується науковометричною базою eLIBRARY.RU.

2. Гусев Е.И. Оптимизация доступа к распределённым страницам памяти в cloud computing системах основанных на shared everything архитектуре используя метод разгрузки очередей / Е.И. Гусев //

Проблеми інформатизації та управління. – 2015. – Том 4, № 52. – С.17-21.

3. Гусев Е.И. Моделирование трафиков и оценка скорости распределённого доступа в системах облачных вычислений с общим ресурсом на примере Oracle RAC/ Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2015. – Вип. 62. – С. 32-42.

Реферується науковометричною базою eLIBRARY.RU.

4. Гусев Е.И. Математическое моделирование распределённой кластерной системы использующей shared everything подход (Oracle RAC) / Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2014. – Вип. 60. – С.106 – 113.

Реферується науковометричною базою eLIBRARY.RU.

5. Гусев Е.И. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций/ Е.И. Гусев, Кулаков А.Ю. // – Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2012. – Вип. 57. – С.76-80.

Реферується науковометричною базою eLIBRARY.RU.

6. Гусев Е.И. Исключение блокирования общего ресурса распределённых системах / Е.И. Гусев // – Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2011. – Вип. 54. – С.162 – 166. – *Здобувачем запропонований алгоритм реалізації розподілених транзакцій, який виключає блокування спільного ресурсу при виконанні та фіксації транзакції.*

Реферується науковометричною базою eLIBRARY.RU.

7. Гусев Е.И. Способы оптимизации совместного доступа к распределённым страницам памяти в системах облачных

вичислений / Е.И. Гусев // матеріали XVI міжнародної наукової конференції IAI -2016 ім. Т.А. Таран 18-20 травня 2016 року. – 2016.

8. Гусев Е.И. Моделирование способов организации доступа к распределённым страницам памяти/ Е.И. Гусев // матеріали XXIII Міжнародної конференції з автоматичного управління «Автоматика 2016» 22-23 вересня 2016 року, м. Суми. – 2016.

АННОТАЦИЯ

Гусев Е.И. Способы организации совместного доступа к распределённым страницам памяти в системах облачных вычислений. – Рукопись.

Диссертация на соискание учёной степени кандидата технических наук по специальности 05.13.05 - Компьютерные системы и компоненты. - Национального технического университета Украины «Киевский политехнический институт имени Игоря Сикорского», Киев, 2017.

Диссертация посвящена проблеме оптимизации работы с общими ресурсами в системах облачных вычислений. В условиях возрастающих требований к объёмам накопления, преобразования и хранения информации, возникает тенденция переноса баз данных в облачную среду. Однако особенности обработки общего ресурса в современных СУБД предъявляют повышенные требования к каналам передачи данных между узлами СУБД по времени отклика. В работе рассмотрены *системы, основанные на репликации изменений,*а также системы, использующие *shared nothing*, *shared disk* или *shared everything* архитектуру. В результате сравнения эффективности масштабирования систем обоснована целесообразность дальнейшего совершенствования способов доступа, базирующихся на *shared everything* архитектуре. В качестве прототипной модели для исследования *shared everything* архитектуры была выбрана система Oracle Real Application Clusters (Oracle RAC), поскольку она имеет ряд существенных преимуществ перед другими известными системами и имеет широкий спектр возможностей, что обеспечивает распространённость системы и влияет на качество её поддержки.

Среди существующих проблем архитектуры рассмотрена перегрузка – ситуация, когда длительность обработки заявки превышает среднюю продолжительность ожидания очереди, образующейся в течение этой

обработки. Обосновано, что перегрузка является наиболее сдерживающим фактором использования глобальных сетей в системах облачных вычислений. Определено, что исследование перегрузок в системах, использующих *shared everything* архитектуру, и разработка способов, позволяющих избежать или значительно снизить вероятность появления перегрузок – наиболее актуально в контексте задачи масштабирования СУБД в системах облачных вычислений.

Для оценки эффективности традиционных способов доступа к распределенным страницам памяти и существующих способов разрешения конфликтов за ресурс автором предложена трёхкомпонентная математическая модель. Первый компонент – модель задержек, описывающая сценарии обработки страниц памяти, необходима для определения продолжительности доступа к страницам, с учётом обеспечения когерентности. Второй компонент – модель конфликтов – отражает реализованный в прототипной системе способ блокирования общего ресурса в контексте взаимосвязей этих ресурсов и транзакций. И третий – модель трафиков – главной целью которой является выделение значимых, с точки зрения исследования распределённых СУБД, классов ресурсов, формализация взаимозависимостей ресурсов и транзакций и определение входных параметров для модели конфликтов и модели задержек.

Предложенная математическая модель помогает определить наиболее перспективные, с точки зрения оптимизации, категории задержек в контексте исследования перегрузок, связанных с конфликтами за общие ресурсы или распределённые страницы памяти, что открывает возможности для создания новых способов доступа к общим ресурсам и распределённым страницам памяти.

В результате анализа с помощью математической модели недостатков существующего способа обработки общего ресурса в работе предложен

способ доступа к общим ресурсам, который по сравнению с традиционным, позволяет исключить перегрузки, снизить требования, как по времени отклика, так и по пропускной способности, к каналам передачи данных при развёртывании СУБД в облачной среде. Применение предлагаемых новаций позволяет рассматривать в качестве транспорта в облачных СУБД не только высокопроизводительные коммутаторы, расположенные внутри одного ЦОДа, но и глобальную сеть, объединяющую ресурсы нескольких ЦОДов, в том числе распределённых географически.

Предлагаемый способ доступа успешно дополняет существующие способы, использующиеся в современных облачных СУБД на базе *shared everything* архитектуры, и предусматривает включение в себя традиционных алгоритмов доступа, успешно зарекомендовавших себя в реализации *shared everything* архитектуры (на примере Oracle RAC). Это позволяет динамически выбирать наиболее адекватный для обрабатываемого трафика алгоритм как из набора традиционных процедур доступа, так и из инновационных, предложенных в работе.

Основной акцент предлагаемых новаций направлен на проблему возникновения перегрузок при доступе к общим ресурсам. В работе обосновывается необходимость сокращения времени отклика для снижения вероятности перегрузки. Для решения проблемы предлагаются два подхода реализованные в трёх методах. Первый подход реализован в методе двухфазного выполнения транзакции и методе конвейеризации фаз. Предлагается использовать версии закешированных в процессе предыдущей обработки страниц при определении списка необходимых общих ресурсов. Использование этих страниц данных – оптимистический подход, поскольку страницы инвалидированы, и, возможно, именно из-за запрашиваемых данных. Но оптимистичность позволяет асинхронно разослать запросы на совместные ресурсы, не организуя цепочку

последовательных обращений, увеличивающих время отклика. Это целесообразно также и потому, что оценка времени, затрачиваемого на определение взаимозависимости транзакций и ресурсов, не намного меньше по сравнению со временем выполнения транзакции. Таким образом, первый раз (первая фаза) в транзакции используются неактуальные страницы. Завершающим этапом метода (второй фазой) является повторное выполнение последовательности операций транзакции с уже полученными актуальными страницами, а затем проверка и исправление ошибок, возникших из-за оптимистического подхода первой фазы, для удовлетворения требований ACID. Разница методов, реализующих подход, в том, что конвейеризация рассмотренных фаз показывает лучшую эффективность транзакций со слабозакешированными ресурсами. «Платой» за выигрыш во времени является более чем двухкратное увеличение объёма операций на каждом узле, что требует определения границ целесообразности применения методов.

Второй подход – метод назначения обработчика ресурса – позволяет уменьшить очереди к страницам, возникающие при высоких интенсивностях обращения. Эта проблема при традиционном (реализованном в Oracle RAC) способе доступа *shared everything* предъявляет высокие требования ко времени отклика между узлами, и, в контексте переноса в облачную среду, является ключевой. Суть метода - зафиксировать для каждой «горячей» страницы узел обработки, минимизировав, таким образом, количество сетевых пересылок при последовательной обработке ресурсов в очереди. Негативным эффектом метода является увеличение объёма межузлового трафика.

В четвёртой главе моделируются три вида трафиков: первые два предложены автором для более полного освещения особенностей предлагаемых новаций, а третий является синтетическим тестом (TPC BENCHMARK Standard Specification Revision 5.11), широко используемым

для определения производительности программно-аппаратных комплексов в ИТ индустрии. Для рассмотренных трафиков проведена экспериментальная проверка предложенной модели и оценка ожидаемого эффекта от применения разработанных процедур доступа и разрешения конфликтов, основанных на предложенных методах. Экспериментально модель сравнивается с реально функционирующим Oracle RAC, что позволяет обосновать её адекватность. Внесение изменений в математическую модель для оценки длительностей обработки предложенных процедур позволяет сравнить эффективность традиционного и модернизированного в этой работе способа доступа к ресурсам и страницам. В течение исследования параметры модели, отвечающие физическим параметрам системы, фиксируются таким образом, чтобы эмулировать глобальную сеть в качестве транспорта. В качестве главной входной метрики для всех трафиков рассматривается средняя интенсивность обращений. В зависимости от распределения классов транзакций внутри трафиков и взаимозависимостей классов ресурсов, составляющих классы транзакций, определяются интенсивности перегрузки, то есть предельные интенсивности, в которых система ещё остаётся в рабочем состоянии, для каждого трафика.

Во всех случаях оценка возможного эффекта была от 35% до 913%. Такой результат создаёт предпосылки для использования глобальных сетей в качестве транспорта, но следует отметить, что характеристики трафика могут в значительной степени повлиять на вероятность перегрузки. Таким образом, предложенный способ может занять заслуженное место в арсенале средств масштабирования облачных систем.

Ключевые слова: Облачные вычисления, СУБД, Oracle RAC, перегрузки, блокировки, очередь, страница памяти, *shared everything* архитектура.

Список публикаций соискателя

1. Гусев Е.И. Моделирование способов организации доступа к распределённым страницам памяти в системах облачных вычислений основанных на shared everything архитектуре / Е.И. Гусев // Вісник НТУУ «КПІ» .Сер. Інформатика, управління та обчислювальна техніка. – 2016. – Вип. 64. – С.133-137.

Реферується науковометричною базою eLIBRARY.RU.

2. Гусев Е.И. Оптимизация доступа к распределённым страницам памяти в cloud computing системах основанных на shared everything архитектуре используя метод разгрузки очередей / Е.И. Гусев // Проблеми інформатизації та управління. – 2015. – Том 4, № 52. – С.17-21.

3. Гусев Е.И. Моделирование трафиков и оценка скорости распределённого доступа в системах облачных вычислений с общим ресурсом на примере Oracle RAC/ Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2015. – Вип. 62. – С. 32-42.

Реферується науковометричною базою eLIBRARY.RU.

4. Гусев Е.И. Математическое моделирование распределённой кластерной системы использующей shared everything подход (Oracle RAC) / Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2014. – Вип. 60. – С.106 – 113.

Реферується науковометричною базою eLIBRARY.RU.

5. Гусев Е.И. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций/ Е.И. Гусев, Кулаков А.Ю. // – Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2012. – Вип. 57. – С.76-80.

Реферується науковометричною базою eLIBRARY.RU.

6. Гусев Е.И. Исключение блокирования общего ресурса распределённых системах / Е.И. Гусев // – Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2011. – Вип. 54. – С.162 – 166. – *Здобувачем запропонований алгоритм реалізації розподілених транзакцій, який виключає блокування спільного ресурсу при виконанні та фіксації транзакції.*

Реферується науковометричною базою eLIBRARY.RU.

7. Гусев Е.И. Способы оптимизации совместного доступа к распределённым страницам памяти в системах облачных вичислений / Е.И. Гусев // матеріали XVI міжнародної наукової конференції IAI -2016 ім. Т.А. Таран 18-20 травня 2016 року. – 2016.
8. Гусев Е.И. Моделирование способов организации доступа к распределённым страницам памяти/ Е.И. Гусев // матеріали XXIII Міжнародної конференції з автоматичного управління «Автоматика 2016» 22-23 вересня 2016 року, м. Суми. – 2016.

ABSTRACT

Gusev E.I. Techniques of organizations shared access to distributed memory pages in cloud computing systems. - Manuscript.

Thesis for a Candidate of Technical Sciences degree in specialty 05.13.05 - Computer systems and components. - National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kiev, 2017.

The thesis is devoted to the problem of shared resource processing optimization in a cloud computing systems. Due to increasing of requirements to information accumulation, conversion, and storage, there is a trend to migrate databases to the cloud. However, shared resource process ability in modern DBMS have high response time requirements during data transferring between nodes in the distributed database. The paper discusses replication-based systems,as well as systems that use *shared nothing*, *shared disk*, or *shared everything* architecture. Comparison of scaling efficiency of systems specifies expediency of further improvement of access methods based on a *shared everything* architecture. As a prototype model for researching the *shared everything* architecture system was chosen Oracle Real Application Clusters (Oracle RAC) because it has significant advantages over other known systems and has a wide range of features that ensures the prevalence of the system.

Among the existing problems of *shared everything* architecture is overloading – a situation when the average duration of waiting queue created during request processing exceeds duration of that request processing. There is proved that overloading is the most limiting factor of global networks usage in cloud computing databases. Determined that the researching of overloading in a *shared everything* architecture systems, and development of techniques to avoid or significantly reduce the probability of overloading is the most relevant in the context of DBMS scaling problems in cloud computing systems.

To evaluate the effectiveness of traditional methods of access to distributed memory pages, and existing methods of resource conflict resolution, the author

proposes a three-component mathematical model. The first component is Delay Model. It describes sceneries of memory pages processing. Delay Model is required to determine page access durations, taking into account coherence provision. The second component is Conflict model. It reflects implemented in the prototype system a method of locking a shared resource in the context of interrelations of these resources and transactions. And the third component is the Traffic model. The main purpose of Traffic model is the identifying of important, from the standpoint of the researching of distributed DBMS, resource classes, formalization of the interdependencies of resources and transactions and the determining of the input parameters for the Conflict model and Delay model.

A mathematical model helps to identify the most promising from the point of view of optimization, categories of delay in the context of the research of overloading-related conflicts for shared resources or distributed memory pages. Delay distribution knowledge opens up opportunities to create new access techniques to shared resources and new approach to distributed memory pages.

As a result of modeling and analysis of traditional techniques weaknesses there is proposed method of access to shared resources, which, in comparison with traditional, allows to eliminate the overload, to reduce the requirements to response time and throughput for data transferring in context of cloud DBMS. The make use of the innovations allows us to to consider as transport in a cloud database not only high-performance switches, located inside one data center, but a global network resources of multiple data centers, including distributed geographically.

The proposed technique successfully complements existing techniques used in modern cloud-based DBMS based on a *shared everything* architecture, and provides the inclusion of traditional, proved as highly-successful, algorithms of access. This allows us dynamically choose the most adequate algorithm for the traffic from a set of traditional and proposed in the thesis procedures for access.

The main focus of the proposed innovations concentrated on the problem of overloading during shared resources access. The work substantiates the necessity of reducing response time to reduce the probability of overloading. As a solution author offers two approaches, implemented in the three methods. The first approach is implemented in the method of two-phase transaction execution (2pe) and in method of phases pipelining. It is proposed to use the version of page cached during previous processing to define a list of required shared resources. The using of these data pages is optimistic approach, because pages are invalidated, and probably because of data, being requested. But approach allows asynchronously send requests for shared resources, without increasing the response time by duration of serial requests. This is acceptably because evaluation time used to determine the interdependence of transaction and resources is small compared to delay of the transaction execution. Thus, for the first time (the first phase) in the transaction is using outdated pages. The final stage of the method (the second phase) is re-execution of transaction sequence of operations with relevant pages already received, and then check and fix errors that occur due to the optimistic approach of the first phase, to satisfy the ACID requirements. Pipelining method compared to 2pe gives better efficiency for transactions with low cached resources. The cost of gain in time is more than twice increase of the volume of processing on each node, which requires the definition of boundaries of appropriateness of the methods.

The second approach – the technique of assigning resource handler– allows us to reduce the queue to the pages that occur at high intensity of treatment. The problem with traditional (implemented in Oracle RAC) access method imposes high requirements to response time between nodes, and, in the context of migration to the cloud, this is key restriction. The essence of the method is to fix processing node for each "hot" page, thus minimize the number of network transfers by sequential processing of the resources in the queue. The negative effect of the method is the increase in inter-node traffic.

In the fourth chapter there are modeled three types of traffics: the first two are suggested by the author for a more complete coverage of the features of the proposed innovations, and the third is a synthetic test (TPC BENCHMARK Standard Specification Revision 5.11), widely used to determine the performance of software and hardware systems in the IT industry. For the considered traffics experimental verification of the proposed model and evaluation of expected effect of the suggested algorithms based on the proposed methods. Experimental model is compared with a real-functioning Oracle RAC to prove its adequacy. To estimate the durations of processing of the proposed algorithms there were making changes to the mathematical model. It allows to compare the performance of traditional and modernized method of access to resources and distributed pages. During the study, parameters corresponding to physical system parameters are fixed so, as to emulate the global network as a transport. As the main input metric for all traffic is considered medium intensity of requests to application. Depending on proportion of transaction classes within the traffic and interdependencies of the resource classes of and transaction classes, in the thesis was determined intensity of the overloading – the maximal intensity in which the system remains in working state for each traffic.

In all cases, the assessment of potential effect has been from 35% to 913%. Such a result creates the preconditions for the use of global networks as a transport, but it should be noted that the characteristics of traffic can greatly affect the probability of overload. Thus, the proposed method can take a well-deserved place in the scaling cloud systems methods.

Keywords: Cloud computing, DBMS, Oracle RAC, overloading, lock, queue, distributed memory page , *shared everything* architecture.

List of candidate's publications

1. Гусев Е.И. Моделирование способов организации доступа к распределённым страницам памяти в системах облачных вычислений основанных на shared everything архитектуре / Е.И. Гусев // Вісник

НТУУ «КПІ» .Сер. Інформатика, управління та обчислювальна техніка. – 2016. – Вип. 64. – С.133-137.

Реферується науковометричною базою eLIBRARY.RU.

2. Гусев Е.И. Оптимизация доступа к распределённым страницам памяти в cloud computing системах основанных на shared everything архитектуре используя метод разгрузки очередей / Е.И. Гусев // Проблеми інформатизації та управління. – 2015. – Том 4, № 52. – С.17-21.

3. Гусев Е.И. Моделирование трафиков и оценка скорости распределённого доступа в системах облачных вычислений с общим ресурсом на примере Oracle RAC/ Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2015. – Вип. 62. – С. 32-42.

Реферується науковометричною базою eLIBRARY.RU.

4. Гусев Е.И. Математическое моделирование распределённой кластерной системы использующей shared everything подход (Oracle RAC) / Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2014. – Вип. 60. – С.106 – 113.

Реферується науковометричною базою eLIBRARY.RU.

5. Гусев Е.И. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций/ Е.И. Гусев, Кулаков А.Ю. // – Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2012. – Вип. 57. – С.76-80.

Реферується науковометричною базою eLIBRARY.RU.

6. Гусев Е.И. Исключение блокирования общего ресурса распределённых системах / Е.И. Гусев // – Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2011. – Вип. 54. – С.162 – 166.

– *Здобувачем запропонований алгоритм реалізації розподілених транзакцій, який виключає блокування спільного ресурсу при виконанні та фіксації транзакції.*

Реферується науковометричною базою eLIBRARY.RU.

7. Гусев Е.И. Способы оптимизации совместного доступа к распределённым страницам памяти в системах облачных вичислений / Е.И. Гусев // матеріали XVI міжнародної наукової конференції IAI -2016 ім. Т.А. Таран 18-20 травня 2016 року. – 2016.
8. Гусев Е.И. Моделирование способов организации доступа к распределённым страницам памяти/ Е.И. Гусев // матеріали XXIII Міжнародної конференції з автоматичного управління «Автоматика 2016» 22-23 вересня 2016 року, м. Суми. – 2016.

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА УМОВНИХ СКОРОЧЕНЬ

БД	База даних
Накат	Застосування журналів, тобто змін до системи, при відновленні БД або її реплікації
Партиціонування	Процес розбиття даних на частини
ПЗ	Програмне забезпечення
СЗД	Системи зберігання даних
СКБД	Система керування базами даних
Шардінг	Процес розподілення даних таблиці по різних вузлах
CAP	Consistency, Availability and Partition Tolerance – характеристики розподілених систем
CR	Consistent Read – цілісне читання: стан розподіленої сторінки
DSS	Decision Suppport Systems – системи підтримки прийняття рішень
GCF	Global Cache Fusion – програмна реалізація корпорації Oracle розподіленої пам'яті
LOB	Large Object – великий об'єкт – структура, що використовується для зберігання неструктурованих даних великого об'єму
MVCC	MultiVersion Concurrency Control – управління

	одночасним паралельним доступом за допомогою багатоверсійності
OLTP	Online transaction processing – онлайнова обробка транзакцій – клас систем, що обробляють достатньо велику кількість коротких транзакцій.
Oracle RAC	Oracle Real Application Clusters – розповсюджена розподілена СКБД
PaaS	Platform as Service – платформа як послуга – модель надання хмарних обчислень
Parsing	Розбір команди інтерпретатором – тобто процес пошуку послідовності команд інтерпретатора, що необхідні для виконання SQL запиту.
PL/SQL	Мова програмування, що використовується у СКБД Oracle
Roll forward	Накат вперед – застосування журналів, тобто змін до системи, при відновленні БД або її реплікації
SaaS	Software as Service – програмне забезпечення як послуга – модель надання хмарних обчислень
SCUR	Shared CURrent – розподілений поточний: стан розподіленої сторінки
Shared disk	Архітектура розподілених обчислень
Shared everything	Архітектура розподілених обчислень
Shared nothing	Архітектура розподілених обчислень

SQL	Structured Query Language – ANSI стандарт популярної мови для взаємодії з СКБД
XCUR	Exclusive CURrent – ексклюзивний поточний: стан розподіленої сторінки

ВСТУП

Сутність розв'язуваної в рамках дисертаційної роботи задачі полягає в оптимізації роботи зі спільними ресурсами в глобальних системах хмарних обчислень. В якості критеріїв оптимізації розглядаються метрики, які мають критичне значення в глобальних мережах, тобто таких мережах, в яких відносно невисокі показники часу відклику і пропускної спроможності. Це, в першу чергу - зниження ймовірності блокування ресурсу, зменшення часу блокування ресурсу і зниження міжвузлового трафіку. В роботі запропонований спосіб доступу до спільних ресурсів, що враховує специфіку обробки в середовищі хмарних обчислень, досліджені моделі трафіків в сучасних базах даних (БД), і проведено порівняння алгоритмів доступу і блокування з існуючими на прикладах досліджуваних моделей.

Актуальність роботи. За більш ніж 50 років історії систем керування базами даних (СКБД) було створено безліч алгоритмів доступу до спільного ресурсу та їх модифікацій. Але з появою хмарних обчислень проблема в СКБД вийшла на інший рівень. Використання існуючих способів блокування в сучасних хмарних системах призводять до значного збільшення часу обробки, і як наслідок, часу блокування. Збільшення часу блокування знижує стійкість системи до перевантажень, пов’язаних з доступом до спільних ресурсів. А це призводить до того, що масштабування методами, що використовувались у розподілених базах даних не забезпечує достатнього рівня ефективності. Результатом сформованої практики є ситуація, коли СКБД в системах хмарних обчислень стає спільним ресурсом, використовуваним усіма обчислювальними потужностями. При масштабуванні хмарної системи збільшення обчислювальних потужностей часто впирається в неможливість пропорційного збільшення потужності (масштабування) СКБД. Проблема вирішується нарощуванням апаратних ресурсів

центральної СКБД або істотною зміною архітектури додатків для зниження інтенсивності звернень до бази даних. Все це обумовлює кризу в БД, у результаті якої з’явилися нові архітектури БД, тісно пов'язані додатками (NoSQL DB) для розвантаження бази даних від запитів, та спостерігається новий виток розвитку дорогих програмно-апаратних комплексів (Oracle Exadata, IMB DB2 PureScale on z/OS). Варто відзначити, що хмарні технології для обчислювальних задач (включаючи рендерінг, трансформацію тощо) привнесли спрощення адміністрування за рахунок консолідації та зниження вартості ресурсів. Однак для СКБД хмарні технології вирішують лише одну задачу – підвищення ефективності управління, в той час як вартість ресурсів, в них задіяних, лишається високою, а у деяких випадках навіть перевищує аналогічні за можливостями традиційні «нехмарні» сервіси. Таким чином, складається ситуація, коли вимоги до потужності при консолідації зростають, а можливості використовувати для консолідації дешеві хмарні технології відсутні. Власне, низькі можливості хмарних систем консолідувати системи зі спільними ресурсами і зумовили кризу в розвитку СКБД в контексті поширення використання систем хмарних обчислень.

Таким чином, тематика дисертаційної роботи, спрямована на розробку ефективних способів доступу до спільних ресурсів в глобальних хмарних системах актуальна, і становить науковий і практичний інтерес.

Зв'язок роботи з науковими програмами, планами, темами.

Робота виконувалася згідно з планами наукових досліджень на кафедрі обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» в рамках науково-дослідних робіт:

- «Розробка теоретичних основ побудови високопродуктивних комп’ютерних систем з динамічним розпаралелюванням обчислювальних процесів» (державний реєстраційний номер

0111U002729 згідно з науковим напрямом «Розробка високопродуктивних багатокластерних обчислювальних систем»);

- «Методи організації моніторингових інформаційно-аналітичних систем науково-освітнього призначення на основ і високопродуктивних обчислювальних кластерних технологій» (державний реєстраційний номер 0109U000526).

Мета і задачі дослідження. Метою роботи є підвищення ефективності способів доступу до спільного ресурсу в системах хмарних обчислень. Для досягнення поставленої мети в роботі вирішуються такі задачі.

1. Аналіз існуючих способів доступу до спільних ресурсів, виявлення їх недоліків і знаходження шляхів можливого удосконалення в середовищі хмарних обчислень для формування математичної моделі системи розподіленої СКБД в контексті вирішення задачі доступу до спільного ресурсу.

2. Розробка математичної моделі процесу одночасного доступу до спільного ресурсу в розподілених СКБД.

3.Розробка способу організації одночасного доступу в розподілених СКБД.

4. Експериментальна перевірка запропонованої моделі, оцінка очікуваного ефекту від застосування запропонованого способу.

Об’єктом досліджень є процес організації сумісного доступу в розподіленій СКБД, яка функціонує в межах системи хмарних обчислень.

Предметом досліджень є способи і засоби блокування спільних ресурсів та оповіщення вузлів в системах розподілених хмарних обчислень.

Методи дослідження ґрунтуються на теорії ймовірностей, теорії множин, використанні методів моделювання та методів комбінаторного аналізу.

Наукова новизна одержаних результатів полягає у вирішенні завдань масштабування в хмарних СКБД, які працюють в середовищі з багатьма користувачами. Запропоновано та обґрунтовано способи, що відрізняються від відомих відходом від класичного блокування при визначенні необхідного набору даних, оптимізацією оповіщення вузлів і асинхронної реалізацією коригуючих дій при вирішенні конфліктів. Такий підхід дозволяє підвищити пропускну спроможність системи хмарних обчислень, знизити час відклику запиту, знизити ймовірність конфліктів, зумовлених блокуванням і підвищити стійкість системи до перевантаження, пов’язаного з блокуваннями.

Автором одержані наступні нові наукові результати.

1. Запропоновано і обґрунтовано математичну модель організації одночасного доступу до спільного ресурсу в системах хмарних обчислень, що дозволяє оцінити ефективність відомих методів доступу і виявити причини, що зумовлюють виникнення перевантаження.

2. Вперше був запропонований і обґрунтований метод двофазного виконання транзакції, який на відміну від відомих методів організації доступу до розподілених сторінок пам’яті, дозволяє виключити блокування спільного ресурсу в системах хмарних обчислень на етапі визначення необхідного набору даних. Це дозволяє істотно збільшити пропускну спроможність систем хмарних обчислень і скоротити час відклику.

3. Запропоновано і обґрунтовано метод призначення обробника ресурсу в системах хмарних обчислень, що дозволяє при виникненні черги знизити кількість послідовних пересилань за рахунок збільшення паралельних. Обґрунтовано доцільність такого підходу і досліджена область застосування запропонованого способу.

4. Запропоновано і обґрунтовано комбінований підхід до вибору процедури доступу до спільного ресурсу, який, на відміну від відомих, дозволяє оптимізувати процедуру спільного доступу в залежності від

інтенсивності транзакцій і кількості оброблюваних ресурсів в кожній з них.

5. Запропоновано спосіб конвеєризації фаз виконання транзакції, який, на відміну від послідовного виконання транзакції, дозволяє поєднати виконання окремих частин транзакції. А це дозволяє знизити ймовірність перевантаження.

Практичне значення одержаних результатів визначається тим, що

1. Запропонований спосіб доступу до спільного ресурсу дозволяє включати в хмарні системи ресурси, не пов'язані з основною хмарою швидким каналом комунікації.
2. Запропоновано модульну математичну модель розподіленої СКБД, що дозволяє моделювати обробку або блокування даних при дослідженні затримок, а також моделювати трафік в залежності від набору операцій і їх розподілу. В силу модульності вона може бути використана і для аналізу інших алгоритмів обробки даних, що реалізуються в хмарних СКБД, в тому числі при дослідженні на стійкість до перевантаження пов’язаного з блокуваннями в залежності від топології, параметрів мережі та трафіку.
3. Проведено аналіз синтетичного тесту продуктивності СКБД TPC-C, що широко застосовується у промисловості, показані слабкі сторони цього тесту при використанні в хмарних системах.

Особистий внесок здобувача. Основні результати отримані автором самостійно. У роботі [6], написаній у співавторстві автором запропонований алгоритм реалізації розподілених транзакцій, який виключає блокування спільного ресурсу при виконанні та фіксації транзакції.

Апробація результатів дисертації.

Основні результати роботи доповідалися і обговорювалися на:

- XVI міжнародній науковій конференції IAI-2016 ім. Т.А.Таран 18-20 травня 2016 року;

- XXIII Міжнародній конференції з автоматичного управління 22-23 вересня 2016 року м. Суми.

Публікації. За результатами виконаних досліджень опубліковано 8 наукових робіт, з яких 6 статей в фахових науково-технічних спеціалізованих виданнях, що включені до переліку наукових фахових видань (з них 5 статей, що входять до баз міжнародної наукової електронної бібліотеки eLIBRARY.RU) та 2 тези доповідей Міжнародних науково-технічних конференцій.

РОЗДІЛ 1. Аналіз існуючих способів доступу до спільних ресурсів, для формування математичної моделі

За більш ніж 50 років історії систем керування базами даних (СКБД) було створено безліч алгоритмів доступу до спільного ресурсу і їх модифікацій. Але з появою хмарних обчислень проблема в СКБД вийшла на інший рівень. Існуючі способи блокування при використанні в хмарних системах призводять до значного збільшення часу обробки і як наслідок часу блокування. Збільшення часу блокування знижує стійкість системи до перевантаження, пов'язаного з обробкою спільних ресурсів [1]. Це автоматично призводить до складнощів масштабування бази даних традиційними методами, які докладніше будуть розглянуті нижче. Результатом сформованої практики є ситуація, коли база даних в системах хмарних обчислень стає спільним ресурсом, який використовується усіма обчислювальними потужностями. При масштабуванні хмарної системи збільшення обчислювальних потужностей часто впирається в неможливість пропорційного збільшення потужності СКБД, тобто у неможливість масштабування. Проблема вирішується нарощуванням апаратних ресурсів центральної бази даних (БД) або істотною зміною архітектури додатків для зниження інтенсивності звернень до БД. Все це обумовлює кризу в СКБД, що виражається як у появі нових архітектур БД, тісно пов'язаних додатками (NoSQL DB [2-7]) для розвантаження БД від запитів, так і новому витку розвитку дорогих програмно-апаратних комплексів (Oracle Exadata [8], IBM DB2 pureScale [9]). Варто відзначити, що хмарні технології для обчислювальних задач (включаючи рендерінг, трансформацію тощо) привнесли і зниження вартості ресурсів, і спрощення адміністрування за рахунок консолідації. Однак для СКБД ними вирішується тільки одна задача – підвищення ефективності управління, на шкоду вартості ресурсів, тобто вимоги до потужності при консолідації

зростають, а можливості використання для консолідації дешевих хмарних технології відсутні. Власне, невисокі можливості хмарних систем консолідувати системи зі спільними ресурсами і зумовили кризу в розвитку СКБД в контексті розвитку систем хмарних обчислень.

1.1. СКБД в контексті розвитку хмарних технологій

Розглядаючи моделі обслуговування хмарних систем, слід зазначити, що надання СКБД як сервіс для клієнта підходить під класифікацію платформа як послуга (Platform as Service, PaaS) [10-12]. Хоча деякі дослідники відносять хмарні СКБД до Software as Service (SaaS) [13] – це, з огляду на визначення дане в [11] – невірно, оскільки у визначенні чітко говориться про «без складності покупки і супроводу програмного забезпечення (ПЗ) та інфраструктури, що лежить в основі сервісу».

Серед компаній, які надають послуги хмарних СКБД, можна виділити такі як: Oracle – на основі власної СКБД [14], Enterprise DB – на основі PostgreSQL [15], Caspio.com – на основі MSSQL [16], ClearDB [17], SkySQL [18] та інші – на основі mySQL. Досить детальна класифікація сьогоднішніх хмарних систем зроблена в роботі [19]. Однак, усі наведені приклади хмарних сервісів орієнтовані на невисокі вимоги обслуговування, а головним плюсом цих сервісів є зниження витрат за рахунок відсутності витрат на володіння сервером. При збільшенні вимог до обслуговування компанії (ті ж Oracle, Microsoft, Enterprise DB) пропонують володіти власними серверами.

Таким чином, всі перелічені рішення є комерційною нішею, а не можливістю збільшувати власну продуктивність (в тому числі в реальному часі) за рахунок використання хмари. Безсумнівно, про що свідчить успішність цих проектів, така ніша необхідна, проте задача збільшення продуктивності за рахунок хмарних систем, в т.ч. у приватній хмарі (private cloud [11]) або корпоративній (community cloud [11]), вони не вирішують.

1.2. Порівняння архітектур хмарних СКБД

Ключовим фактором при побудові хмарних систем є вибір архітектури. На сьогоднішній день виділяють такі основні архітектури: *shared nothing* [20], *shared disk* [21], *shared everything* [22], а також рішення *на основі асинхронної або синхронної реплікації*, приклади яких будуть розглянуті нижче. Крім того, незважаючи на явну неефективність з точки зору продуктивності, в деяких випадках використовуються системи, які використовують розподілені транзакції [23]. Наведемо переваги і недоліки кожної з цих архітектур на прикладі існуючих реалізацій.

Shared nothing. Прикладом успішних реалізацій цих підходів є mysql NDB cluster [24], ElasTras [25], Teradata [26], xkoto Gridscale [27]. Основною проблемою у реляційній моделі для *shared nothing* є необхідність частого доступу до відносно невеликого набору даних [19]. Зауважимо також, що «невеликий» по відношенню до спільного обсягу не означає невеликий з точки зору витрат на обробку, хоча існують трафіки, для яких ця проблема незначна. Можна навести приклади обходу даної проблеми – *shared something*, який засобами програми або ін'єкції (наприклад тригером) [19] поділяє запит на частини згруповані за логічним принципом. Однак замість одного запиту в багатьох випадках ми отримуємо стільки, скільки вузлів в нашій системі. Тобто такий підхід добре себе зарекомендує в випадках коли витрати на вибірку великі (тобто аналітичні запити до сховища) і буде вкрай неефективним в разі коли запит повертає лише кілька рядків. Компанія Teradata чітко позиціонує свою розподілену систему як сховище [26]. Крім того, неефективне глобальне індексування та з'єднання декількох таблиць. Щодо глобального індексування, то після обробки індексу немає гарантії, що необхідні дані знаходяться на тому ж, а не на віддаленому вузлі. Проблеми індексів та з'єднань будуть розглянуті далі більш детально. Резюмуючи вищесказане про *shared nothing* можна говорити про невисоку

ефективність використання такої системи для обробки трафіку OLTP (online transaction processing)[58].

Shared disk. Ця архітектура не дуже часто використовується в СКБД. Серед прикладів можна привести Oracle Parallel Server [28], систему, яка вже багато років не підтримується. Причина досить проста – для синхронізації дані треба записати на спільний диск і прочитати звідти. Оскільки швидкості звернення до дискових пристроїв (час відклику) нижче ніж звернення до пам'яті, в наступних версіях корпорація Oracle перейшла на архітектуру *shared everything*. Також *shared disk* архітектура згідно [28] використовується і у хмарних рішеннях на основі IBM DB2, хоча використання технології pureScale [9] переводить ці системи в категорію *shared everything*. Варто зазначити, що для систем з невеликою кількістю спільних ресурсів така архітектура через свою простоту може бути корисна. Ще один недолік цієї архітектури – наявність файлових систем, які коректно працюють з розподіленим між декількома вузлами диском і проблема когерентності закешованих даних.

Резюмуємо – так як і для *shared nothing* можна констатувати невисоку придатність до обробки OLTP трафіку.

Рішення на основі реплікації. *Рішення на основі реплікації* як синхронної, так і асинхронної дуже широко поширені. Можна згадати standby сервери, на які з основної бази виноситься read-only підтрафік (наприклад, пов'язаний зі звітністю) [30] або так званий read-mostly підтрафік, що містить переважно читання, але крім того дуже рідко виконує запис [30]. Крім того, варто згадати системи вибіркової реплікації: Oracle Golden Gate [31], Quest SharePlex [32], а також системи, засновані на застосуванні (часто фільтрованих) бінарних журналів mySQL. Однією з найбільш вдалих систем на основі застосування бінарних журналів є Percona XtraDB Cluster від компанії Percona [33]. Однак, незважаючи на широке поширення всі перераховані системи мають один недолік: на кожен

вузол необхідно реплікувати зміни з усіх інших вузлів. Це накладає жорстке обмеження: сумарний обсяг змін всіх систем не повинен перевищувати можливості самого повільного вузла на застосування змін (т. з. «накат» або «roll forward»). На практиці такі системи рідко містять число вузлів більше трьох саме через це обмеження. Що стосується систем, які використовують розподілені транзакції (наприклад [39]), то вони є окремим випадком синхронної реплікації. Можна було б багато говорити про неефективність найбільш поширених алгоритмів 2PL [34] та 2PC [35], але не вони є слабким місцем систем, які використовують їх, а сама архітектура на основі реплікації. Можливо, саме тому модернізовані алгоритми блокування і фіксації транзакції [36,37,38] не знаходять широкого застосування в контексті зростання популярності хмарних систем.

Все це спонукає звернути увагу на архітектуру *shared everything*, реалізовану в Oracle Real Application Clusters (Oracle RAC) [40], яка порівнюючи з вищезгаданими архітектурами дозволяє масштабувати систему:

- без явних обмежень на співвідношення читання і запису в порівнянні з архітектурою *на основі взаємної реплікації*;
- має незаперечну перевагу в швидкодії у порівнянні з архітектурою *shared disk*;
- не має проблем масштабування архітектури *shared nothing*;

Проте необхідно вказати і на обмеження архітектури *shared everything*. В першу чергу – це високі вимоги до часу відклику між вузлами. Наслідком цього є той факт, що глобальна мережа як міжвузловий транспорт може використовуватися тільки за умови істотних обмежень на трафік, які, як буде показано в цій роботі, накладають обмеження на інтенсивність зміни спільних ресурсів. Причому інтенсивність змін обмежена не співвідношенням змін та читань в спільному трафіку при

заданій кількості вузлів (як в архітектурі *на основі взаємної реплікації*), а безпосередньо часом відклику мережі.

Оскільки архітектура *shared everything*, як показано вище, має найкращі можливості для обробки OLTP трафіку, система Oracle RAC була обрана в якості прототипної моделі для дослідження. Крім того, Oracle RAC має широкий спектр впроваджень, що забезпечує поширеність системи і впливає на якість її супроводу, що також є перевагою для дослідження.

1.3. Оглядове дослідження хмарних СКБД в контексті обробки спільного ресурсу

На жаль, тема спільного ресурсу, незважаючи на те, що вона ключова, в роботах, що стосуються хмарних СКБД не розглядається детально [41, 19, 47] або взагалі ігнорується [43, 45, 46, 48-50]. У деяких роботах [42, 51] ігнорування проблеми спільного ресурсу взагалі ставить питання про доцільність досягнутих результатів, а в інших [44] – істотно обмежує сферу застосування запропонованих новацій. Нижче буде приведена докладний аналіз зазначених робіт. Однак в цілому, резюмуючи роботи [42-46, 48-51] варто вказати на зосередженість на технічних реалізаціях *shared nothing* архітектур і ігнорування ключового значення індексів для OLTP систем, а також факту, що безпосередньо на вибірку і модифікацію даних при обробці OLTP [57-58] запитів витрачатися багато менше часу, ніж на сумарну тривалість таких затримок як: обслуговування структур управління спільним ресурсом, мережева взаємодія, реалізація цілісності, безпеки, забезпечення розбір (parsing) запиту та інших затримок, що виникають при обробці запиту. Багато в чому такий розподіл затримок став причиною популяризації noSQL СКБД згаданих раніше, хоча не тільки він. Розглянемо зазначені помилки докладніше:

Основна ідея в роботах [44], [42], [51] – це можливість на рівні

додатку або сервера додатків визначити той вузол хмарної системи, який містить потрібні дані. Далі в роботах [44], [51] автори зосереджуються на задачах, що і яким чином розподілити по вузлах, декларуючи різні метрики критеріями якості масштабування. І дійсно, бувають трафіки, для яких рівномірність розподілу [51] або вибір об'єктів для шардінга (sharding) [80] (для мінімізації з'єднань таблиць) [19,44] визначають ефективність масштабування. Але тоді потрібно точно визначити, для яких трафіків це можливо. У разі хмарної СКБД, це означає вивчення інженерами хмарного провайдера взаємозв'язків ресурсів в транзакціях перед тим, як підтвердити можливість розміщення СКБД в хмарному середовищі. Що досить дивно для PaaS [11]. Коли це дійсно можливо? Наприклад в ситуації оновлення інформації про абонента або формування виписки по абоненту, якщо кожному вузлу призначається свій список абонентів, що не перетинається з вузлами, як і пропонується в роботі [44]. Назвемо трафік, що задовольняє таким умовам, «абонентським». По суті шардінг – це партиціонування [52] з партиціями рознесеними на різні вузли. Варіантами може бути як горизонтальне так і вертикальне партиціонування, однак проблеми в усіх випадках будуть ідентичні. Для прикладу з роботи [44], вже пошуковий запит з фільтром за датою оплати, трансформується в n запитів до всіх вузлів, де n – кількість вузлів.

Важливо відзначити, що якщо відсоток часу на власне вибірку даних в загальному часу обробки запиту досить великий, то від розпаралелювання такого запиту ефект буде, в іншому випадку – ні. Як наслідок, навантаження на систему кратно збільшується. Таким чином можна констатувати, що ефективність такого партиціонування безпосередньо залежить від співвідношення «виграшних» запитів до «програшних».

«Виграшні» – це:

- запити з ключем партиціонування в умовах пошуку
- важкі запити без з'єднання або з з'єднанням з партиції тільки свого

вузла, в яких вибірка даних переважає над іншими витратами на виконання;

«Програшні» – це:

- запити повертають невелику кількість рядків і не містять порівняння на рівність з ключем партиціонування серед умов фільтру;
- важкі запити, але на відміну від виграшних важких запитів, що містять після вибірки даних порівняння (не обов'язково заради з'єднання) з даними, що обираються з інших вузлів. Прикладом програшного важкого запиту для наведеної структури даних може бути найбільші 5 боржників за кожен місяць;
- запити з з'єднанням (при з'єднанні вийшли за межі вузла);

Варто зауважити, що запит «найбільші 5 боржників» не завжди буде повільніше, ніж якби він виконувався б на 1 вузлі, але з точки зору споживаних ресурсів ми отримуємо програш, що в кінцевому рахунку негативно позначається на масштабуванні. Такий запит буде тим ефективніше, чим більше витрати на власне вибірку в загальних витратах на виконання запиту. Аналогічно зі з'єднаннями таблиць, в яких при великій кількості вузлів необхідно розглядати всі комбінації партицій різних таблиць [4]. У даній роботі ми не будемо зупинятися на проблемі ефективності шардінга [4], відзначимо лише те, що наведена проблема є особливістю *shared nothing* архітектури.

Крім того, *shared nothing* архітектура накладає обмеження на індекси. Існування глобального індексу [54] по всім партиціям неефективно, оскільки, при розташуванні на одному вузлі цей вузол може стати вузьким місцем під час масштабування. Якщо глобальний індекс задублювати на кожному вузлі – то його оновлення призведе до кратного кількості вузлів збільшення навантаження. Підхід, незважаючи на витрату ресурсів, може бути прийнятним, за умови, що інтенсивність оновлень глобального індексу низька. Це характерно для багатьох трафіків, але вимагає аналізу

запитів трафіку для прийняття такого рішення. Якщо ж глобальний індекс «розмазати» по партиції – то при запитах нам буде необхідно обробляти частини одного запиту на кількох вузлах. Постановка задачі не дає однозначної відповіді буде ефективно це чи ні – але можна констатувати збільшення часу обробки запиту, а у випадках коли витрати на власне вибірку невеликі, відносна вага непродуктивної роботи по міжвузловій взаємодії теж різко зростає. Також очевидно, що тимчасові витрати на такий підхід будуть завжди більше, ніж у відповідній *shared everything* конфігурації, оскільки при наведеному підході («розмазаний» глобальний індекс) пересилання в *shared nothing* здійснюється завжди, а в *shared everything* тільки, якщо дані змінилися. Крім того, всі недоліки *shared everything* – а це вимоги до часу відклику між вузлами в такій конфігурації успадковуються *shared nothing* глобальним індексом.

Зупинимось на неефективності при обробці з'єднань [4] у *shared nothing*. Запити з з'єднанням – відома проблема при шардінгу, яка вирішується різними способами [55], [56]. Сутність проблеми – це необхідність з'єднання всіх партицій однієї таблиці з усіма партиціями іншої. У деяких випадках є можливість забезпечити розташування партицій, що об'єднуються, на одному вузлі [42], [44], але в загальному випадку *shared nothing* неефективний при таких запитах. У той же час *shared everything* при масштабуванні демонструє результати, де залежність від кількості вузлів не носить степеневого характеру, що буде зрозуміло після розгляду сценаріїв доступу в розділі 2.

Таким чином, ефективність масштабування *shared nothing* систем залежить від характеру трафіку, правильного вибору способу масштабування і вимагає істотної модернізації алгоритмів з'єднання таблиць. І хоча алгоритми з'єднання таблиць удосконалюються [56], в найбільш популярних СКБД, які використовують *shared nothing* архітектуру як основу для побудови хмари (MS SQL, MySQL, PostgreSQL) ці рішення не реалізовані.

Варто зауважити, що трансформація СКБД у хмарну, для забезпечення PaaS [11], зажадає динамічної розбивки ресурсів на партиції (sharding) в результаті аналізу трафіку. Основні ж *shared nothing* реалізації розподілених систем вживають статичний підхід і не можуть розглядатися як PaaS, тому що вимагають від користувача оптимізації способів розбиття на партиції з прив'язкою до архітектури.

Оцінюючи *shared something* [19], можна сказати, що по суть підходу – за рахунок правильного розташування даних знизити ймовірність програшних запитів у порівнянні з виграшними. Тобто забезпечити придатність хмари для трафіку. Це знову ж таки впирається в необхідність для задоволення вимог PaaS забезпечити автоматичне «розмазування» ресурсів по вузлах системи хмарних обчислень для мінімізації неефективності шардінга, що на жаль автором ідеї не виконується.

Завершуючи огляд хмарних СКБД варто зупиниться на огляді хмарних систем для noSQL. Відзначимо, що розглянутий вище так званий «абонентський» трафік добре масштабується в хмарних системах. Згадаємо також і те, що трафіки соціальних мереж, здебільшого, також абонентські. І тут необхідно звернути увагу на дуже цікавий факт, який змінює місцями, в контексті вивчення критеріїв трансформації СКБД в хмарну, причину і наслідок: noSQL СКБД можуть ефективно обслуговувати тільки «абонентський» трафік. Це означає, що оскільки вони були використані при розробці, такі системи легко автоматично розділяти за значеннями

первинного ключу на партиції, і при цьому у запитах міжпартиційна взаємодія не має сильного впливу на загальну продуктивність роботи. Тобто задача приміщення таких завдань в хмару, яка задовольняє вимогам PaaS – тривіальна. Що ми і спостерігаємо на практиці. З інженерів хмарного провайдера автоматично знімається задача аналізу придатності трафіку для хмари, організованого на *shared nothing* архітектурі. Критерієм придатності для розміщення в хмарі стає використання noSQL СКБД. Вибір розробником noSQL при проектуванні програми автоматично виконує роботу за інженера хмарного провайдера.

Коротенько розглянемо банківські системи. Ситуація, що з одним рахунком працюють одночасно 2 сесії – малоймовірна. Отже, маємо ознаки абонентського трафіку. Однак причина непопулярності noSQL в цьому сегменті інша: високі вимоги до збереження даних, забезпечуються вимогами ACID [62]. В роботі [41] автор приділяє достатньо уваги значенню ACID для збереження даних, і розвінчанню маніпуляцій популяризаторів noSQL [67], [2-7], більшість з яких випливають з сумнівною «теореми» CAP [63], та її доказу [64]. Наводячи аргументи [65], [66] автор успішно доводить актуальність вимог ACID для забезпечення збереження даних, в тому числі і при високій продуктивності. Зауважимо, що задача визначення «абонентського трафіку», і критеріїв ефективності розміщення його даних в хмарі вимагає більш чіткої формалізації і виходить за рамки цієї роботи, але при дослідженні хмарних СКБД повз такого факту проходити не можна. Також потрібно відзначити, що обраний в якості прототипу Oracle RAC порушує «теорему» CAP [63], оскільки виключення вузла з кластера, через відмову мережі, що є випадком «partition tolerance», яке вказуються в «теоремі», не приводить до втрати доступності (*availability*) або узгодженості (*consistency*). І на завершення, хотілося б взагалі виділити зі списку літератури роботу [41], яка хоч і не

присвячена проблемі спільного ресурсу, але чітко фокусується на актуальних проблемах хмарних СКБД.

1.4. Що впливає на ефективність масштабування хмарних СКБД, але виходить за рамки дослідження

Робота буде сфокусована на оптимізації доступу до спільного ресурсу і сторінок пам'яті, в хмарному середовищі. Сферою дослідження стануть процедури доступу до спільного ресурсу, що застосовуються на рівні вузлів обробки в оперативній пам'яті. Зауважимо, що оптимізація організації сторінки пам'яті (такі техніки як зберігання в стовбцях або рядках, кластеризація в рамках однієї сторінки і вибір, які дані будуть зберігатися разом на одній сторінці, змінний розмір сторінки, використання не сторінкової, а ресурсної (функціональної) організації пам'яті тощо) також може представляти науковий сенс, однак також виходить за рамки дослідження. Ми в роботі будемо вважати, що використовується зберігання в рядках, оскільки в найбільш поширеній системі, де використовує *shared everything* підхід (Oracle RAC, процедури доступу якої будуть розглянуті в розділі 2 в якості прототипу), застосовується такий спосіб зберігання [40].

Крім того, робота не стосується оптимізації власне зберігання даних. Незважаючи на те, що Oracle RAC використовує *shared disk* як частину *shared everything* підходу, в розподіленій хмарній системі швидкості доступу до сховищ даних можуть бути різними. З урахуванням відмінностей бізнес-вимог по продуктивності, надійності і надмірності з'являється досить великий простір для оптимізації зберігання даних. Цей факт представляє науковий інтерес, проте також в роботі розглянуто не буде. У роботі ми будемо вважати, що швидкість доступу до всіх сховищ (хмарне сховище) однакова з усіх вузлів. З практичної точки зору, це допустимо, оскільки широко застосовується в існуючих рішеннях. Наприклад, в прототипі дослідження – технології Oracle RAC використовується тільки проста топологія зв'язків всіх вузлів з усіма

системами зберігання даних (СЗД) за допомогою високопродуктивних каналів зв'язку. Більш того, в контексті оптимізації доступу до спільного ресурсу ми будемо вважати, що розмежування доступу до спільного ресурсу має бути повністю вирішено на рівні глобального кешу сторінок (Global Cache Fusion – GCF) [86]. Також будемо вважати, що звернення до сховищ для читання даних настільки рідкісні, що часовими витратами на них можна знехтувати, оскільки вся активна частина бази закешована в глобальному кеші. Це справедливо для OLTP систем (on line transaction processing) [57-58]. Часом фіксації транзакції, вважаючи, що фіксація – це гарантування збереження даних на постійний носій, ми також будемо нехтувати, оскільки існує можливість відпустити блокування перш, ніж почати виконання операції фіксації (commit).

Також може бути цікавою і сам процес фіксації транзакції в хмарній СКБД. Це:

- алгоритми взаємодії з блокуваннями;
- методи забезпечення відновлення зафіксованих транзакцій (в т.ч. в реальному часі на інших вузлах);
- різні правила визначають зафіксовані даних, що відображають бізнес вимоги щодо безпеки даних і функціональні особливості систем;

Наприклад, ми можемо вважати транзакцію зафіксованої тоді, коли зміни транзакції задубльовані в хоча б у трьох локальних вузлах з двох різних сегментів мережі, і при цьому не вимагати негайного скидання змін на СЗД, оптимізуючи таким чином ввід-вивід на СЗД і значно підвищуючи час відклику фіксації. Тут відкривається величезний простір для оптимізації, особливо з огляду на те, що надмірність даних на різних вузлах може з'являтися в результаті рішення задач оптимізації доступу до спільного ресурсу неявно, як наслідок виконання цих завдань. Але, незважаючи на перспективність питання констатуємо, що оптимізація процесу фіксації транзакції також виходить за рамки даного дослідження.

На завершення відзначимо, що в даній роботі ми не будемо розглядати розбір (parse) запиту та його оптимізацію, однак звернемо увагу на те, що тема оптимізації запитів в хмарних СКБД також становить значний науковий інтерес.

1.5. Основні недоліки *shared everything* архітектури

Реалізація в Oracle RAC *shared everything* підходу незважаючи на високу ефективність, підтверджену поширеністю системи [59] має низку недоліків. Перерахуємо проблеми, які є у систем, що використовують класичний *shared everything* підхід, після чого реалізуємо методи, які їх усувають. У четвертому розділі ми промоделювавши кілька трафіків покажемо ефект від запропонованих новацій і більш чітко окреслимо сферу застосування, де запропоновані методи є найбільш ефективними.

1.5.1. Проблема наростаючої черги

Проблема наростаючої черги була розглянута в [60]. Суть проблеми в наступному. Припустимо, ми маємо систему, на яку надходять заявки з інтенсивністю *v*. Система має деяку множину спільних ресурсів. Обробка заявки займає час *t*. Якщо заявка, що надійшла в систему, звернувшись до спільного ресурсу знаходить, що спільний ресурс зайнятий – вона очікує його звільнення. Якщо більше однієї заявки очікує спільний ресурс – доступ до ресурсу здійснюється в порядку черги FIFO. Можна розкласти спільний потік заявок на *n* непересічних потоків, кожен до свого ресурсу і зі своєю інтенсивністю.

Позначимо інтенсивність заявок до максимально ресурсу, який частіше всього запитується, як $v_{конфл}$. Будемо вважати, що процес стаціонарний, тобто ймовірність приходу того чи іншого числа заявок за час довжиною Δt залежить тільки від довжини цієї ділянки і не залежить від його розташування на осі часу протягом певного проміжку часу $\Delta\tau$ ($\Delta t << \Delta\tau$). Тоді $v_{конфл} = const$, і потік пуасонівський протягом цього $\Delta\tau$.

Нехай час обробки 1й заявки – *t*. Якщо утворилася черга з 2 заявок, то

час обробки другий заявки це t плюс час стояння в черзі. (Позначимо t_x, причому t_x менше t, тому що друга заявка надійшла після першої): Якщо черга з 3х заявок – то аналогічно – час обробки 3й заявки t_x+2t. Якщо утворилася черга з n заявок, то час обробки n-ої заявки: $t_x+(n-1)t$, час блокування ресурсу – в такому випадку – $t_x+(n-1)t$, за умови що за час $t_x+(n-1)t$ заявок до цього ресурсу не буде. Якщо за час $t_x+(n-1)t$ до ресурсу надійде менше ніж n заявок – то черга не збільшиться, а якщо більше – то збільшиться. Якщо черга збільшиться – то час блокування ресурсу після проходження часу $t_x+(n-1)t$ також збільшиться.

Для зручності аналізу виділимо три стану системи:

1. Стаціонарне – $\nu_{конфл}<1/t$ і ймовірність перевищення інтенсивністю $\nu_{конфл}$ величини 1/t невелика. Черги (більше однієї заявки) не утворюється;

2. Перехідний – $\nu_{конфл}<1/t$ але ймовірністю перевищення інтенсивністю $\nu_{конфл}$ величини 1/t нехтувати не можна: в системі утворюються черги довжиною більше однієї заявки;

3. Перевантаження – $\nu_{конфл}<1/t$. Черга збільшується і час обробки кожної наступної заявки, що приходить в чергу, більший, ніж попередньої. Якщо утворилася черга більше однієї заявки і $\nu_{конфл}<1/t$ – то черга починає скорочуватися, але час обробки заявки в цьому випадку все одно буде включати очікування в черзі тобто буде більше, ніж власне час витрачається на обробку. Дослідження проблем освіти та скорочення черг - перехідні процеси - представляють науковий інтерес, проте виходять за рамки питань цієї роботи. Ми зосередимося на завданні одержання середньої довжини черги.

Таким чином, можна констатувати кілька висновків:

1. Зменшуючи час блокування при обробці заявки, ми при більшій інтенсивності конфлікту залишимося в робочому стані.

2. У перехідному стані можливе істотне (n кратне) збільшення часу відклику в разі утворення черги.

3. Пікове короткочасне збільшення інтенсивності здатне створити чергу, що позначиться на часі відклику наступних заявок протягом ще деякого часу після завершення піку. Зменшуючи час обробки заявки, ми знижуємо ймовірність збільшення часу відклику при виникненні піку. Таким чином, в контексті зниження часу блокування актуальною стає задача розробки таких способів організації доступу до спільного ресурсу, щоб проблема наростаючої черги була виключена, або хоча б час блокування ресурсу не наближатися до граничної інтенсивності переходу в перевантаження.

Завершуючи зауважимо, що висновки, які ми зробили, будуть справедливі не тільки для пуасонівського потоку. Головна вимога – монотонність зростання ймовірності приходу заявки від часу очікування.

1.5.2. Проблема наростаючої черги в контексті багаторесурсної транзакції

У сучасних СКБД ми досить часто маємо справу зі складними транзакціями, тобто такими, що оброблюють велику кількість ресурсів. Це підтверджується наборами операцій, які використовуються в сучасних популярних тестах продуктивності СКБД [68-72]. У багатьох випадках кількість ресурсів може визначатися не тільки і не стільки бізнес вимогами, але і процесом розробки програми. Наприклад, при купівлі товару необхідно змінити стан складу, внести зміну в журнали безпеки, сформувати заявку на доставку, призначити заявку на доставку одного з кур'єрів (з урахуванням оптимізації маршруту кур'єра), і нарахувати бонуси за покупку в особистому профілі клієнта. Всі дії крім заповнення журналу безпеки не вимагають входження в ту ж транзакцію, що і зміна стану складу, і можуть бути виконані асинхронно. Але часто розробник їх включає в ту ж транзакцію, керуючись міркуваннями продуктивності або простоти проектування. В результаті, продовжуючи наведений приклад в

одній транзакції ми оперуємо п'ятьма таблицями. Якщо припустити що на кожну таблицю буде одночасно обслуговуватися три індекси, то отримуємо ситуацію оновлення 20 ресурсів під час однієї транзакції. У розділі буде показано, що в разі конфлікту за спільні сторінки в системі, яка використовує *shared everything* підхід, буде виконуватись від 3 до 8 пересилань. В середньому за рахунок кешування трохи менше – але необхідно зазначити, що ефект кешування невеликий на багатовузлових системах.

Таким чином, час обробки транзакції для вищенаведеного прикладу, при відсутності конфліктів буде наближатися до часу від 60 до 160 пересилань. Цей факт висуває дуже серйозні вимоги до часу пересилання, оскільки, як було показано в розділі 1.5.1, при наближенні інтенсивності надходження заявок цього ресурсу до величини, оберненої часу суми цих пересилань, підвищується ймовірність перевантаження при зверненні до критичних ресурсів. Проблема актуальна для будь-якої складної транзакції, яка одночасно оброблює велику кількість ресурсів. Зауважимо, що в багатьох випадках неможливо виділити обробку частини ресурсів в окремий асинхронний процес. Крім того, часто можна зіткнутися зі складною структурою об'єкту, оптимізованої під інші задачі, що також збільшує кількість ресурсів задіяних у транзакції.

Резюмуючи позначену проблему потрібно відзначити, що для сучасних систем, які використовують *shared everything* архітектуру, (на прикладі Oracle RAC), проблема вирішується збільшенням швидкості інтерконекту, результатом якого є зниження часу відклику мережі, по якій здійснюється обмін повідомленнями. Однак такий підхід виключає використання ресурсів пов'язаних між собою глобальною мережею, де і час відклику мережі менше, і розкид цього часу досить великий. Таким чином стає актуальною задача оптимізації алгоритмів доступу до ресурсів для роботи в глобальних мережах.

1.6. Постановка задачі в контексті виконаного огляду

На основі аналізу вимог використання, можливостей хмарних СКБД і огляду архітектур, в контексті проблеми спільного ресурсу в середовищі хмарних обчислень, для OLTP трафіку можна резюмувати:

- Незважаючи на те, що спроб реалізовувати хмарні СКБД на основі *shared nothing* робиться найбільше, архітектура має серйозне обмеження: трансформація СКБД в хмарну, здійсненна не для всіх трафіків. Це суперечить вимогам PaaS [11].
- Невідповідність вимогам ACID виключає зі списку перспективних розробок клас СКБД noSQL, що використовують *shared nothing* архітектуру. NoSQL – це нішове рішення [41].
- Масштабування *рішень на основі реплікації* обмежено співвідношенням читань і змін ресурсів в трафіку.
- *Shared everything* увібрала в себе всі переваги *shared disk*, і мінімізувала вплив недоліків, що означає, що *shared everything* перевершила технологію, яка передувала собі .

З огляду на вищевикладене, архітектура *shared everything* виглядає перспективніше інших, що дозволяє зробити припущення, що усунення її проблем є також найбільш перспективним в контексті розвитку хмарних СКБД. Оскільки у якості прототипу *shared everything* доцільно розглядати систему Oracle RAC, а для неї обмеженням масштабованості є проблема наростаючої черги, дослідження роботи варто зосередити на цій проблемі. Для цього необхідно вивчити існуючі способи обробки спільних ресурсів в контексті обслуговування багаторесурсної транзакції і запропонувати їх модернізацію або реорганізацію.

1.7. Висновки по розділу

1. Розглянуто тенденції розвитку баз даних з урахуванням

особливостей функціонування у хмарному середовищі. В результаті порівняння хмарних СКБД з традиційними виявлено, що хмарні системи характеризуються більш високими вимогами до рівня масштабування.

2. В контексті викликів хмарних технологій розглянуто основні архітектури СКБД: *shared nothing*, *shared disk*, *рішення на основі реплікації* та *shared everything*. Проведено порівняння цих архітектур, виявлені загальні переваги та недоліки кожної із них.

3. Проведено аналіз наукових праць, присвячених функціонуванню СКБД у хмарному середовищі, на основі якого виявлено ігнорування проблеми спільного ресурсу, та зосередженість на деталях реалізацій нішових рішень, де значення проблеми незначне. Такий підхід не має серйозних перспектив розвитку, оскільки ігноруються вимоги концепції «платформа як сервіс» (PaaS), що вимагає зосередитись на удосконаленні способів обробки спільного ресурсу у системах хмарних обчислень.

4. Базуючись на аналізі існуючих систем, у якості прототипу для дослідження було обрано систему Oracle RAC, оскільки для неї властивий необхідний рівень зрілості та використовується найбільш перспективна, в контексті функціонування у хмарному середовищі, архітектура *shared everything*.

5. В результаті аналізу основних недоліків *shared everything* архітектури, проблема наростаючою черги, при обробці багаторесурсних транзакцій, визначена основною причиною, яка стримує масштабування.

РОЗДІЛ 2. Математична модель процесу одночасного доступу до спільного ресурсу в розподілених базах даних

Використання традиційних способів блокування в системах хмарних обчислень призводить до значного збільшення часу обробки запиту і, як наслідок, часу блокування. Така ситуація призводить до неможливості масштабування хмарної СКБД при використанні класичних способів блокування спільного ресурсу без істотного збільшення швидкості міжвузлового обміну [1]. Реалізована в Oracle RAC *shared everything* архітектура використовує найбільш перспективний на сьогодні спосіб обробки спільних сторінок пам'яті в хмарних СКДБ при обслуговуванні OLTP трафіку.

Для виявлення проблем підходу *shared everything* пропонується використання моделі, що враховує специфіку обробки сторінки. При моделюванні *shared everything* архітектури буде використана трикомпонентна модель, що складається з моделі затримок [61], моделі трафіків [75] і моделі конфліктів. Структурно модель затримок реалізує особливості апаратної реалізації, модель трафіків (ймовірнісна модель) особливості функціонування, тобто ймовірностей проходження тих чи інших сценаріїв обробки в залежності від трафіку. А модель конфліктів визначає структуру конфліктів і дозволяє оцінити тривалості конфліктів в залежності від реалізації і функціонування системи. Виходячи з того, що система повинна функціонувати з різними трафіками і того, що можливості зміни апаратної реалізації системи обмежені – найбільш перспективним, з точки зору модернізації, є зміна процедури доступу до сторінки, яка знаходиться переважно в фокусі моделі конфліктів, а також в моделі затримок. Для вирішення задачі модернізації способу доступу до розподіленої сторінки досліджуються основні затримки, що вносяться існуючими процедурами доступу, виходячи з моделі конфліктів. Оскільки є можливість зняти часові затримки з реально функціонуючого Oracle Real

Application Clusters (Oracle RAC), доцільно оцінити адекватність запропонованої моделі, порівнявши її з реально функціонуючим Oracle RAC.

2.1. Модель затримок обробки

Ключовою особливістю Oracle RAC є *shared everything* архітектура [40]. Це передбачає як спільний диск, так і спільну оперативну пам'ять на всіх вузлах. Однак, відразу зауважимо, що «спільна пам'ять» – в даному випадку – умовний термін. На практиці, кожен вузол має власну оперативну пам'ять, але здійснюється пересилання сторінки (блоку) даних від одного вузла до іншого за запитом, якщо сторінка (блок) закешована на іншому вузлі. При такій організації виникає задача відстеження стану сторінок на всіх вузлах: а саме: яка сторінка на якому вузлі закешована, і на який момент часу. Задача вирішується введенням вузла власника (майстер-вузол) для кожної сторінки. Всі вузли будуть слати запити на зміну стану сторінки до цього майстер-вузлу. Стан сторінки може бути XCUR (eXclusive CURrent – ексклюзивний поточний), SCUR (Shared CURrent – розподілений поточний), CR (Consitent Read – цілісне читання – тобто читання на деякий момент часу в минулому, за умови дотримання транзакційної цілісності даних, які читаються) і PI (Past Image exist – після змін, зручний при відновленні, найчастіше таку дію конвертує в CR) [40]. Пояснимо стани: якщо ми збираємося змінювати сторінку – ми запитуємо у майстер-вузла блокування на сторінку XCUR. Якщо збираємося читати найостаннішу версію сторінки – то SCUR. Майстер-вузол відстежує, яким вузлам видані SCUR блокування, а яким XCUR. Не більше однієї копії будь-якої сторінки може бути закешовано в XCUR стані в кожен момент часу. На відміну від XCUR – SCUR версії сторінки можуть бути присутніми на всіх вузлах. Перед модифікацією сторінки, процес повинен отримати на неї блокування XCUR, що викликає переведення всіх SCUR сторінок на всіх вузлах в CR стан. CR передбачає, що блок актуальний на

деякий момент часу в минулому, але на даний момент існує пізніша версія блоку. PI – стан в якому залишається блок після стану XCUR, тобто інший вузол запросив XCUR або SCUR блокування, і вузол, раніше утримує XCUR, її віддав. Сторінка виходить в стані PI, який згодом може бути конвертований в CR. Важливо сказати, що одна і та ж сама сторінка може одночасно перебувати на різних вузлах, в своєму стані для кожного вузла. З точки зору дослідження, важливо виділити 5 станів сторінки, з урахуванням наявності її різних версій і станів кожної версії на кожному вузлі в глобальному кеші:

1. XCUR + *n* * CR; *n* – ціле, невід'ємне число. У цьому стані на одному вузлі на одній з версій сторінки XCUR блокування, а на інших вузлах – версії сторінки в CR. Кількість CR старих версій на кожному вузлі будь-яка, стільки, скільки версій сторінки знадобилося протягом функціонування вузла, причому, версії в CR станах можуть бути і на вузлі, де утримується XCUR блокування. Позначимо стан *xcur* або *x*.

2. *k* * SCUR + *n* * CR; *k* – натуральне число, *n* – ціле, невід'ємне число. На кожному вузлі не більше однієї версії з SCUR блокуванням, але хоча б одна версія блоку в стані SCUR існує на будь-якому з вузлів. Одночасно з цим, у CR станах на всіх вузлах, може бути необмежена кількість версій сторінки. Позначимо стан *scur*.

3. *n* * CR (XCUR або SCUR вилетів), *n* – ціле, невід'ємне число. У глобальному кеші не залишилося XCUR або SCUR версій блоку. Варто зауважити, що при першому зверненні до сторінки SCUR або XCUR запитом вона буде прочитана з диска для обробки, після чого залишиться в стані SCUR або XCUR в залежності від запиту. Однак, при запиті CR стану (на певний момент часу) – необхідності зачитувати з диска блок немає, якщо блок на потрібний момент часу залишився закешованим. У даній праці розглянуто класичний GCF алгоритм [77], що дозволяє при повторному запиті не збирати блок, звертаючись до *undo* сегментів [77], а

використовувати, у разі існування, раніше зібрану локально закешовану версію, якщо необхідні моменти часу співпадають. Крім класичного підходу в якості технік оптимізації може бути запропонована техніка пошуку CR блоку в глобальному кеші. Якщо майстер-вузол зберігає історію отримання XCUR блокувань і історію CR запитів до неї, вона може вказати на якому вузлі існують вже зібрані на потрібний момент CR блоки. Але, як було сказано вище, в цій праці розглядається класичний підхід. Позначимо цей стан «cr only».

4. Блоку немає в глобальному кеші. Позначимо стан «ні».

5. Майстер-вузол при зверненні до іншого вузла виявляє, що раніше закешована в стані SCUR або XCUR версія сторінки вилетіла. Позначимо стан «вилетів».

Залежно від того, закешована сторінка на поточному вузлі і який запит (читання або запис) був попереднім, змінюється процедура доступу до сторінки. Назвемо елементарні послідовності дій, які реалізують доступ до сторінки в залежності від вхідних умов, сценаріями. Знаючи ймовірності вхідних умов і тривалості сценаріїв, можна обчислити середній час доступу до сторінки, як суму добутків тривалості сценарію на його ймовірність. Ймовірності вхідних умов визначаються моделлю трафіків, а тривалості сценаріїв – моделлю затримок. Розпишемо існуючі сценарії.

Як вказувалось в праці [90] – сценарії різні у випадках, коли не існує закешована в глобальному кеші блоку, існує один або кілька закешованих сторінок з SCUR блокуванням і вже існує закешована сторінка з XCUR блокуванням на будь-якому з вузлів. Таких сценаріїв дев'ять. Розпишемо ці сценарії:

Сценарій 1. У разі якщо сторінка незакешована ні на одному з вузлів (про те, що сторінка незакешована, майстер-вузлу відомо) маємо таку ситуацію:

1. Запит XCUR / SCUR блокування у майстер-вузла

2. Відповідь майстер-вузла про відсутність сторінки в глобальному кеші.

3. Читання сторінки зі спільного диску.

Позначимо його як «читання з диску» («read from disk», rd1). Розпишемо тривалість затримки для сценарію:

$$t_{rd\,1} = 2t_{net} + t_{io} \quad (2.1)$$

де: t_{io} – час читання блоку з диску; t_{net} – середній час пересилання службового мережевого пакету між вузлами;

Сценарій 2. Якщо ж майстер-вузол має інформацію про закешовану сторінку, але при зверненні виявляється, що сторінка вилетіла з кешу, то сценарій такий:

1. Запит XCUR / SCUR блокування у майстер-вузла
2. Відправлення вузлу, де закешована сторінка, інструкції «переслати сторінку вузлу, що її запитав».
3. Відповідь вузла, де сторінка вилетіла з кешу, майстер-вузлу про відсутність її в кеші.
4. Відповідь майстер-вузла вузлу, що запитав сторінку, про її відсутність в глобальному кеші.
5. Читання сторінки зі спільного диску.

Позначимо його як «читання з диску 2» («read from disk 2», rd2):

$$t_{rd\,2} = 4t_{net} + t_{io} \quad (2.2)$$

Сценарій 3. Якщо ж майстер-вузол має інформацію про закешовану сторінку і ця сторінка в глобальному кеші в стані SCUR, маємо:

1. Запит XCUR блокування у майстер-вузла
2. Майстер-вузол відсилає всім вузлам, на яких ця сторінка в стані SCUR, повідомлення, що надійшов запит на зміну блоку (XCUR), і з якого вузла.
3. Стан сторінки на вузлах, де був SCUR, змінюється на CR.
4. Один з вузлів пересилає вузлу, що запросив XCUR саму сторінку.

5. Вузол, який захопив XCUR блокування, підтверджує майстер-вузлу її захоплення.

Позначимо сценарій як «запис-читання» («write-read», wr) і розпишемо затримки. Важливим є те, що обробка наступного ресурсу в сесії може не чекати завершення сценарію, а починається одночасно з кроком 6. Тоді затримки:

$$t_{wr} = 2t_{net} + t_{send} \quad (2.3)$$

де: t_{send} – час пересилки сторінки пам'яті від одного вузла до іншого.

Сценарій 4. Якщо ж майстер-вузол має інформацію про закешовану сторінку і вона в глобальному кеші в стані XCUR, маємо:

1. Запит XCUR блокування у майстер-вузла
2. Майстер-вузол відсилає вузлу, у якого цей блок в стані XCUR, повідомлення, що надійшов запит на зміну блоку (XCUR) і з якого вузла.
3. Очікування завершення обробки (існує черга модифікувати сторінку)
4. Стан блоку на вузлі, де був XCUR, змінюється на PI.
5. Вузол, де був XCUR, пересилає вузлу, запити XCUR, саму сторінку.
6. Вузол, який захопив XCUR блокування, підтверджує майстер-вузлу її захоплення.

Позначимо сценарій як «запис-читання» («write-write», ww). Відзначимо, що обробка наступного ресурсу в сесії може не чекати завершення сценарію, а починатися одночасно з кроком 6. Маємо:

$$t_{ww} = 2t_{net} + t_{send} \quad (2.4)$$

Аналогічно розпишемо 2 сценарії отримання SCUR блокування при наявності блоку в глобальному кеші. Різниця між ними в тому, в якому стані (SCUR або XCUR) закешована сторінка.

Сценарій 5. Випадок, коли на одному або декількох вузлах закешована версія сторінки з блокуванням SCUR:

1. Запит SCUR блокування у майстер-вузла.
2. Пересилання майстер-вузлом вузлу, на якому закешована сторінка зі SCUR блокуванням, інструкції «переслати поточну версію сторінки вузлу», що її запитував у майстер-вузла.
3. Пересилання сторінки зі SCUR блокуванням вузлу, що її запитував.
4. Пересилання підтвердження отримання блокування майстер-вузлу.

Позначимо цей сценарій як «читання-читання» («read-read», rr), також відзначимо, що обробка починається одночасно з останнім кроком.

$$t_{rr} = 2\,t_{net} + t_{send} \quad (2.5)$$

Сценарій 5. Якщо існує версія сторінки з XCUR блокуванням на будь-якому з вузлів, то послідовність дій така:

1. Запит SCUR блокування у майстер-вузла.
2. Пересилання майстер-вузлом вузла, на якому закешована сторінка з XCUR блокуванням, інструкції переслати поточну версію сторінки вузлу, що запросив її.
3. Очікування завершення обробки (при виникненні черги на модифікацію сторінки).
4. Пересилання сторінки вузлу, що її запитував. При цьому XCUR блокування може бути конвертована (знижена) в SCUR, і тоді вузлу, який запитував SCUR стан, сторінка в стані SCUR і буде надана. Інший варіант – блокування залишиться XCUR, а запит вузлу буде віддана сторінка в стані CR – але на поточний момент часу. Рішення про зниження блокування приймається на другому кроці сценарію майстер-вузлом. На жаль, алгоритм прийняття рішення про зниження блокування не розкривається корпорацією Oracle. Можна запропонувати, наприклад, що знижується блокування до SCUR в разі повторного запиту SCUR блокування. Ми при моделюванні черг будемо вважати, що блокування знижується завжди. З точки зору часових затримок сценарію час доступу залишається

незмінним і не залежить від зниження блокування при виконанні сценарію. Алгоритм прийняття рішення про зниження може бути предметом оптимізації, але в даній праці це питання не розглядається.

5. Пересилання підтвердження захоплення блокування майстер-вузлу.

Позначимо цей сценарій як «читання-запис» («read-write», rw), також, як в сценарії 4, обробка ресурсу в сесії може не чекати завершення сценарію, а починається одночасно з кроком 5.

$$t_{rw} = 2\,t_{net} + t_{send} \quad (2.6)$$

У системі існує також сценарій «запис на диск» («write-disk» wd). Але з точки зору аналізу часу відклику або сумарних затримок у всіх сесіях, сценарій не цікавий, оскільки відкладений запис на диск здійснюється фоновим процесом DBWn асинхронно в ґрунтуючись на накопиченні змінених блоків в кешах вузлів. Оскільки затримки фонових процесів [79] ми не класифікуємо як частину часу сесій користувачів, час, витрачений на фоновий запис на диск, у формулу затримок не увійде.

Ще 2 сценарії – це запити CR стану, відтворення якого здійснюється при забезпеченні мультиверсійного читання (*MVCC - MultiVersion Concurrency Control*) [76, 77]. В даному дослідженні ми будемо вважати, що запит CR стану може бути перетворений або в запит декількох SCUR блоків, або, якщо сторінка, для якої запитаний момент часу збігається з часом копії, збереженої в кеші, в сценарій читання готового блоку з локального (або глобального) кешу. Таким чином, маємо сценарії:

Сценарій 8 (оптимізований): потрібний CR блок на потрібний момент часу виявився в кеші (Позначимо сценарій як crp *)

Сценарій 9: Потрібного CR блоку не виявилося в кеші і довелося його збирати. (Позначимо сценарій як crp).

Розглянемо 9й сценарій (crp): Якщо в локальному кеші немає необхідного блоку на запитаний момент часу, то для реконструкції блоку до

моменту часу в минулому повинні бути задіяні блоки *undo* сегментів [76,77]. Система виконує 5 кроків:

1. Отримання SCUR блокування на сторінку, що змінюється
2. Читання сторінки для визначення задіяних у зміні блоку транзакцій.
3. Визначення undo сегментів, задіяних незавершеними транзакціями
4. Отримання SCUR блокувань на сторінки цих undo сегментів.
5. Застосування undo інформації для відновлення блоку на необхідний момент часу.

Фактично запитується стан SCUR як на необхідну сторінку, так і ще на одну або кілька сторінок undo – тобто запитується кілька SCUR блоків. Збільшення кількості оброблюваних блоків, а саме отримання на більшу кількість блоків SCUR блокування, негативно позначається на ефективності системи. Однак ймовірність каскадного запиту блоків undo зумовлена тільки характером трафіку. Будемо вважати, в контексті моделювання Global Cache Fusion (GCF) [77], що характеристиками трафіку є співвідношення запитів SCUR, XCUR і CR *, а значить трафік з великим відсотком блоків undo, що каскадно запитуються, з точки зору моделі, що розглядається в цій праці – це трафік, в якому більший відсоток SCUR запитів в порівнянні з XCUR і CR *.

Сукупність часів обробки, що відповідають розглянутим сценаріям і є моделлю затримок. Розпишемо в табличній формі сценарії доступу до блоку, описані вище, і розставимо відповідні 8 сценаріїв – без сценарію wd, оскільки до часу доступу до блоку в призначених для користувача сесіях він не має відношення, а виконується фоновим процесом DBWn.

Символом «-» в таблиці позначені заборонені стани, тобто запит таких станів неможливий, що зумовлено існуючою процедурою доступу до сторінки в глобальному кеші [77], яка була вище розглянута.

Таблиця 2.1

Розподіл сценаріїв в залежності від стану сторінки та запитів

		Існує у глобальному кеші				ні
		xcur	scur	cr only	вилетів	
Запит стану	cr*	crp*			-	-
	xcur	ww	wr	-	rd2	rd1
	scur	rw	rr	-		

2.1.2. Аналіз затримок часу у різних сценаріях доступу до сторінки

Слід вказати на те, що фактори локального кешу і локального майстер-вузла істотно скорочують час обробки. Наявність сторінки в локальному (для вузла який виконує обробку сторінки) кеші в потрібному стані призводить до того, що в сценаріях ww, rr, wr і rw не потрібно пересилати саму сторінку. З іншого боку, якщо вузол, що обробляє запит, є майстер-вузлом – то не потрібно запитувати стан сторінки по мережі.

Розпишемо таблицю затримок з урахуванням цих умов. За основу візьмемо табличну форму алгоритму, додавши в таблицю 2.1 чинники локального майстер-вузла (вузол обробки є майстер-вузлом для оброблюваної сторінки) і локального кешу (сторінка виявилася локально закешованою).

Варто вказати, що оскільки ми досліджуємо хмарні системи, які оброблюють OLTP трафік, стан даних при читанні і внесенні змін до сторінки має бути на поточний (current) момент часу. Це означає, що сторінки будуть запитуватися для читання в SCUR, а для запису в XCUR стані. Зауважимо, що якщо в системі немає довгих запитів тоді в ній відсутні crp * і cr трафіки. Забігаючи вперед відзначимо, що в досліджуваних в розділі 4 трафіки, довгих запитів немає. Крім того, в контексті дослідження конфліктів при використанні мультиверсійного читання [76,77] не має значення обсяг підтрафіка crp * і cr в загальному

трафіку. Відзначимо, що в прототипній моделі Oracle RAC мультиверсійне читання використовується завжди.

Таблиця 2.2

Розподіл сценаріїв в залежності від стану сторінки та запитів враховуючи фактор майстер-вузлу.

	Запит стану	Існує в глобальному кеші і в якому стані					вилетів	ні
		xcur		scur		cr only		
		локально	нелокал.	локально	нелокал.			
майстер	cr*	crp*					rd2	rd1
	xcur	ww	ww	wr	wr	-		
	scur	rw	rw	rr	rr	-		
Не майстер	cr*	crp*					rd2	rd1
	xcur	ww	ww	wr	wr	-		
	scur	rw	rw	rr	rr	-		

Ще ми будемо вважати, що читань з диска не відбувається, оскільки вся активна частина БД закешована. Таке припущення цілком обґрунтовано, з огляду на невеликий обсяг активно змінюваних даних в сучасних системах в порівнянні з доступним об'ємом пам'яті. Більш того, завдання адміністратора БД часто і полягає в тому, щоб дані, що активно використовуються читалися з кешу, а не з диску. Природно, для систем звітності (DSS) [78], а також змішаних OLTP-DSS систем ця вимога ускладнює реалізацію, але для OLTP – це справедливо. Тоді трафік складається фактично з 4 сценаріїв: rr, rw, wr і ww. З урахуванням факторів локального майстер-вузла і локального кешу маємо 16 сценаріїв. Відмінності, що вносяться цими факторами, досить прості: в разі якщо вузол виявився майстром для сторінки, що потрібна – звернення до майстра здійснюється без мережевого пересилання, а фактор локального кешу дозволяє використовувати наявну в локальному кеші сторінку, якщо запросили стан, що запрошується не перевищує наявний (тобто з XCUR в

XCUR, зі SCUR в SCUR та з XCUR в SCUR), і мінімізувати до двох пересилань до майстра і назад у разі підвищення стану (з SCUR в XCUR). Також відзначимо, що після читань ми не очікуємо черги, а після запису – очікування можливе. Тоді час t_{basic} без урахування очікування черги і факторів локального кешування і майстер-вузла:

$$t_{basic}=t_{ww}=t_{rw}=t_{wr}=t_{rr}=2t_{net}+t_{send} \quad (2.7)$$

З урахуванням усього вищезгаданого маємо таблиці 2.3 та 2.4, що відоб-

Таблиця 2.3

Розподіл тривалостей сценаріїв (скорочений) в залежності від стану сторінки та запитів (без затримок MVCC та затримок звернення до диску)

Запрос состоя-ния		Існує в глобальному кеші і в якому стані			
		xcur		scur	
		л	г	л	г
майстер	xcur	0	$t_{net}+t_{send}$	0	$t_{net}+t_{send}$
	scur	0	$t_{net}+t_{send}$	0	$t_{net}+t_{send}$
Не майст.	xcur	0	$2t_{net}+t_{send}$	$2t_{net}$	$2t_{net}+t_{send}$
	scur	0	$2t_{net}+t_{send}$	0	$2t_{net}+t_{send}$

Таблиця 2.4

Розподіл ймовірностей сценаріїв (скорочений) в залежності від стану сторінки та запитів (без затримок MVCC та затримок звернення до диску)

Запит стану		Існує в глобальному кеші і в якому стані			
		xcur		scur	
		л	г	л	г
майстер	xcur	$1/n\ p_{xx}(1-\varphi)$	$1/n\ p_{xx}\varphi$	$1/n\ p_{sx}(1-\varphi)$	$1/n\ p_{sx}\varphi$
	scur	$1/n\ px_{xs}(1-\varphi)$	$1/n\ p_{xs}\varphi$	$1/n\ p_{ss}(1-\varphi)$	$1/n\ p_{ss}\varphi$
немайст.	xcur	$(1-1/n)\ p_{xx}(1-\varphi)$	$(1-1/n)p_{xx}\varphi$	$(1-1/n)\ p_{sx}(1-\varphi)$	$(1-1/n)p_{sx}\varphi$
	scur	$(1-1/n)\ p_{xs}(1-(1-\varphi)$	$(1-1/n)p_{xs}\varphi$	$(1-1/n)\ p_{ss}(1-\varphi)$	$(1-1/n)p_{ss}\varphi$

ражають тривалості та ймовірності сценаріїв без урахування затримок для забезпечення мультиверсійного читання та запитів до диску. Зауважимо також, що довжину черги ми розглянемо в розділі 2.2.1, а ймовірність незакешування (позначимо її φ) в розділі 2.2.2. Ймовірність того, що вузол буде майстром, враховуючи те, що в системі n вузлів, і вони однакові за фізичними характеристиками ми оцінили в формулах таблиці 2.4 як $1/n$.

Знаючи тривалості сценаріїв і розподіл їх ймовірностей нескладно вивести середній час доступу до сторінки:

$$t_{acc.0}=(p_{xx}+p_{sx}+p_{xs}+p_{ss})\varphi(\frac{1}{n}(t_{net}+t_{send})+(1-\frac{1}{n})(2t_{net}+t_{send}))+p_{sx}(1-\varphi)(1-\frac{1}{n})2t_{net}=$$
$$=\varphi(\frac{1}{n}(t_{net}+t_{send})+(1-\frac{1}{n})(2t_{net}+t_{send}))+p_{sx}(1-\varphi)(1-\frac{1}{n})2t_{net}=$$
$$=\varphi(2t_{net}+t_{send}-\frac{t_{net}}{n})+\frac{\nu_r}{(\nu_r+\nu_w)}\frac{\nu_w}{(\nu_r+\nu_w)}(1-\varphi)(1-\frac{1}{n})2t_{net}=$$
$$=\varphi(2t_{net}+t_{send}-\frac{t_{net}}{n})+2\frac{\nu_r\nu_w}{(\nu_r+\nu_w)^2}(1-\varphi)(1-\frac{1}{n})t_{net}$$

для зручності позначимо коефіцієнт, що передує t_{net} символом β:

$$\beta=2\frac{\nu_r\nu_w}{(\nu_r+\nu_w)^2}(1-\varphi)(1-\frac{1}{n}) \quad (2.7.1)$$

Тоді:

$$t_{acc.0}=\varphi(2t_{net}+t_{send}-\frac{t_{net}}{n})+\beta t_{net} \quad (2.7.2)$$

коефіцієнт β залежить тільки від кількості вузлів і співвідношення між інтенсивностями читання та запису. Нижче ми покажемо, що φ також залежить тільки від кількості вузлів і співвідношення між згаданими інтенсивностями. Оскільки для кожного ресурсу співвідношення читання і запису різне – коефіцієнти β і φ можуть набувати різних значень, але для ресурсів з однаковим співвідношенням читання і запису значення $t_{acc.0}$ – також однакові. Виникає необхідність визначити поняття клас ресурсу. Введемо його більш строго, орієнтуючись не на співвідношення інтенсивностей читання і запису, а орієнтуючись на рівність пари (ν_r,ν_w) . Таким чином, два ресурси належать одному класу ресурсів, коли кожна з інтенсивностей (ν_r,ν_w) першого ресурсу рівна

відповідним інтенсивностям другого. Тоді можна вважати, що ресурси, які відповідають сутностям [73] одного типу (наприклад рядки однієї таблиці, входи одного індексу, LOB дані, що відповідають одному атрибутові [74]) відносяться до одного класу ресурсів.

Тоді для *i*-того класу ресурсів:

$$t_{acc.0.i} = \varphi_i \left(2 t_{net} + t_{send} - \frac{t_{net}}{n}\right) + \beta_i t_{net} \quad (2.7.3)$$

Необхідно відзначити важливість поняття клас ресурсів, оскільки тривалості очікування черг безпосередньо залежать від того, які класи ресурсів обробляються.

2.2. Модель конфліктів

Важливо відзначити очевидний факт, що конфлікт за спільний ресурс в сучасних системах виникає тільки якщо ресурс змінюється. А оскільки ми визначаємо систему як транзакційну, то всі зміни можуть виконуватися тільки всередині транзакцій. Відзначимо ще один очевидний факт, що якщо немає одночасно виконуваних транзакцій, то немає і конфлікту за спільний ресурс. З іншого боку, одночасні транзакції можуть не стикатися за спільний ресурс. Які транзакції будуть конфліктувати, а які ні – визначається трафіком.

Складність моделювання конфлікту в системі, що використовує *shared everything* архітектуру, в тому, що існують два рівня конфлікту. Перший – конфлікт вузлів безпосередньо за сторінку пам'яті, а другий – це конфлікт безпосередньо за ресурс на сторінці (рядок таблиці, індексний вхід, *undo* вхід і т.д.).

Оскільки ми знехтували можливістю читання з диска, припускаємо, що в трафіку немає cr * і crp * підтрафіків, і, як результат, ми фокусуємося на на чотирьох основних сценаріях (ww, rw, wr і rr), де виникає спільний ресурс «сторінка», робота з яким викладена у

відповідних чотирьох сценаріях. Важливо відзначити слідства застосовуваних в сценаріях алгоритмів:

- Обробка складається з отримання блокування, можливого пересилання сторінки і, безпосередньо, читання або запису даних. Наведена послідовність в обробці не може бути порушена.
- Обробка сторінки може бути читанням ресурсу зі сторінки (вимагає SCUR блокування) або записом (вимагає XCUR блокування) Блокування XCUR старше SCUR – тобто, якщо вузол має XCUR блокування – він може виконувати також і читання.
- XCUR блокуванням в системі в будь-який момент часу може володіти лише один вузол, в той час як SCUR – будь-яка підмножина наявних вузлів. Отримання XCUR будь-яким з вузлів, інвалідує всі (і XCUR і SCUR) блокування, на всіх інших вузлах. Отримання SCUR будь-яким з вузлів, інвалідує тільки XCUR блокування якщо вона була на якомусь вузлі, і не змінює статус наявних SCUR блокувань.
- Обробка сторінки здійснюється кожним з вузлів в порядку приходу запиту на обробку в загальну систему, тобто використовуючи FIFO.
- SCUR і XCUR блокування на один ресурс взаємовиключні в рамках всієї системи, тобто, якщо хоча б на одному вузлі існує SCUR блокування на певну сторінку – то XCUR не може бути на жодному вузлі і навпаки: якщо на одному з вузлів існує XCUR блокування на певну сторінку – то SCUR не може бути на жодному вузлі.

Відзначимо, що блокування на сторінку, що видно з алгоритмів, які лежать в основі сценаріїв, передається разом зі сторінкою, тобто, блокування утримується до закінчення обробки сторінки. Причому, навіть у разі, якщо обробка не може бути виконана – наприклад ресурс, який необхідно було змінити виявився заблокованим, замість внесення змін до сторінки, надсилається запит в чергу до ресурсу, а сторінка віддається наступному, в черзі FIFO на сторінку, вузлу.

Другий конфлікт безпосередньо за ресурс (наприклад ексклюзивно за рядок) має своє блокування – блокування за ресурс. Цей конфлікт, на відміну від конфлікту за сторінку, утримує блокування за ресурс до кінця транзакції. Як згадувалось вище, при виявленні на сторінці заблокованого ресурсу (наприклад рядка), система відправляє запит майстру цього ресурсу на постановку в чергу FIFO до ресурсу, який відстежує завершення транзакцій і постачає інструкції вузлам, що виконують транзакції, в яких вказано, яким наступним вузлам, і на які ресурси потрібно передати блокування після завершення транзакції. Спосіб блокування нагадує спосіб запиту сторінки, з тією різницею, що завершення транзакції може викликати масову передачу блокувань ресурсів, задіяних у транзакції різним вузлам. Механізм призначення власника черги (майстра) до ресурсу буде розглянуто нижче. Також необхідно відзначити, що блокування ресурсів завжди тільки XCUR – тобто ексклюзивні.

Розглянемо детальніше ці два рівня блокування вважаючи потік вхідних заявок стаціонарним потоком, і обчислимо, відповідно, середні часи обробки як сторінки (з урахуванням часу перебування в черзі на сторінку), так і транзакції (з урахуванням і часу обробки сторінок кожного класу ресурсу, що входить в транзакцію, і очікування в черзі до цього ресурсу).

2.2.1. Конфлікт за сторінки

Критична, з точки зору проблеми спільного ресурсу, ділянка в сценаріях обробки сторінки починається з передачі самої сторінки в сценаріях, які потребують XCUR блокування ww та rw. Це означає, що можна виконати перші 2 пересилання (тривалістю $2t_{net}$: звернення до майстер-вузлу і відправлення майстер-вузлом повідомлення вузлу, який встановив ексклюзивне блокування, команди віддачі сторінки вузлу, який запросив блокування) паралельно з очікуванням черги спільною сторінки.

Таким чином, на час очікування черги на сторінку перші два пересилання не впливають, а крок черги буде t_{send}, тобто при одному конфлікті – довжина черги становить один зсув – t_{send}; при двох – $2t_{send}$, при n – відповідно $n \cdot t_{\text{send}}$.

Зауважимо, що в сценаріях, які вимагають SCUR блокування, передача сторінок зі SCUR блокуванням може здійснюватися одночасно декільком вузлам. Тут варто зазначити особливості реалізації, які нам насправді компанія розробник не розкриває. Але ми будемо вважати, в разі сценарію wr (XCUR => SCUR), що поки майстер-вузлу не прийшло підтвердження про отримання SCUR блокування вузлом її запит, розсилкою сторінок з SCUR блокуванням займається вузол, який до цього утримував XCUR.

Тоді можна констатувати, що інтенсивність зміни сторінки (позначимо її v_{pl}; pl – page locking)), що впливає на довжину черги – це інтенсивність запису в сторінку. За умови, що потік заявок пуасонівський з інтенсивністю зміни v_{pl}, розрахуємо ймовірності конфліктів і відобразимо їх разом з відповідними тривалостями. Кількість конфліктів від одного і до нескінченості відобразимо в першому рядку таблиці 2.4, у другому рядку відобразимо тривалість конфлікту, а в третьому – ймовірність такої події:

Таблиця 2.5

Кількості конфліктів, та відповідні кількостям тривалості та ймовірності

1	*2*		*k*
t_{send}	$2t_{send}$	...	kt_{send}
$\left(v_{pl} t_{wpl}\right) e^{-v_{pl} t_{wpl}}$	$\frac{\left(v_{pl} t_{wpl}\right)^2}{2!} e^{-v_{pl} t_{wpl}}$	...	$\frac{\left(v_{pl} t_{wpl}\right)^k}{k!} e^{-v_{pl} t_{wpl}}$

Час t_{wpl} – це час, протягом якого проводиться спостереження. Чи не складно обчислити середній час додаткового очікування черги $t_{p.queue}$ як суму ряду:

$$t_{p.queue} = \sum_{k=1}^{\infty} t_{send} \frac{\left(\nu_{pl} t_{wtl}\right)^k}{k!} e^{-\nu_{pl} t_{wpl}} = \nu_{pl} t_{wpl} t_{send} \quad (2.8)$$

Однак варто звернути увагу на один істотний нюанс. Перша заявка не обов'язково чекає повний час t_{send}. Вона буде у діапазоні [0, t_{send}]. З огляду на пуасонівський характер потоку, середній час, в разі одного конфлікту, буде $t_{send}/2$, тобто на $t_{send}/2$ менше. Аналогічно, і для інших значень конфліктів, усі тривалості будуть на $t_{send}/2$ менші, що відображено в табл. 2.6:

Таблиця 2.6

Кількості конфліктів, та відповідні кількостям тривалості та ймовірності (ефект першого конфлікту)

1	*2*		*n*
$t_{send} - t_{send}/2$	$2t_{send} - t_{send}/2$	...	$nt_{send} - t_{send}/2$
$\left(\nu_{pl} t_{wpl}\right) e^{-\nu_{pl} t_{wpl}}$	$\frac{\left(\nu_{pl} t_{wpl}\right)^2}{2!} e^{-\nu_{pl} t_{wpl}}$	...	$\frac{\left(\nu_{pl} t_{wpl}\right)^n}{n!} e^{-\nu_{pl} t_{wpl}}$

$$t_{p.queue} = \nu_{pl} t_{wpl} t_{send} - \frac{t_{send}}{2}\left(1 - e^{-\nu_{pl} t_{wpl}}\right) \quad (2.9)$$

З формули 2.9 видно, що час очікування черги до сторінки $t_{p.queue}$ залежить від часу спостереження t_{wpl} . З іншого боку нас цікавить скільки заявок прийде за час $t_{send}/2$, разом з часом очікування черги, тобто:

$$t_{wpl} = t_{send} + t_{p.queue} = t_{send} + \nu_{pl} t_{wpl} t_{send} - \frac{t_{send}}{2}\left(1 - e^{-\nu_{pl} t_{wpl}}\right) \quad 2.10$$

Таким чином, розв'язавши рівняння ми зможемо знайти t_{wpl}, після чого віднімаючи з нього t_{send} – знайти $t_{p.queue}$. Одним із способів вирішення рівняння можна вважати ітераційний, що відображає фізичні особливості нашої моделі, а саме: розпишемо відомі факти в систему з двох рівнянь:

$$\begin{cases} t_{wpl} = t_{send} + t_{p.queue} \\ t_{p.queue} = \nu_{pl} t_{wpl} t_{send} - \frac{t_{send}}{2}\left(1 - e^{-\nu_{pl} t_{wpl}}\right) \end{cases} \quad (2.11)$$

На першій ітерації візьмемо початкове значення $t_{p.queue}=0$ і підставимо його в перше рівняння системи. Далі отримане значення t_{wpl} можна

підставити в друге рівняння системи і отримати $t_{p.queue}$. Потім, отримане $t_{p.queue}$ – в перше рівняння і так далі. Ітерації можна продовжувати до тих пір, поки різниця отриманих на сусідніх ітераціях значень не стане менше необхідної точності, або до тих пір, поки $t_{p.queue}$ не перевищить t_{send}. Друга умова є критерієм переходу в перевантаження, яке було розглянуто в розділі 1.5.1. На даному етапі зауважимо, що умова перевищення $t_{p.queue}$ значення $t_{p.send}$ говорить про неможливість функціонування системи в штатному режимі при заданій інтенсивності ν_{pl}, а рівняння 2.10 в цьому випадку не має сенсу оскільки втрачає зв'язок з об'єктом дослідження. Зауважимо, що інтенсивність ν_{pl} дорівнює інтенсивності запису в сторінку, а з урахуванням кешування маємо:

$$\nu_{pl} = \varphi \nu_{w} \quad (2.11.1)$$

Також завершуючи питання тривалості черги до сторінок хотілося б вивести загальну формулу доступу до сторінки, включаючи чергу. Позначимо цю тривалість t_{acc} використовуючи формулу 2.7.3 отримуємо:

$$t_{acc.i} = t_{acc.0.i} + t_{p.queue.i} = \varphi_i \left(2 t_{net} + t_{send} - \frac{t_{net}}{n}\right) + \beta_i t_{net} + p_{queue.i} \quad (2.11.2)$$

2.2.2. Оцінка ефекту кешування.

При отриманні системи рівнянь (2.11) ми вважали, що крок черги – t_{send}. Однак існує ймовірність, що заявки, які оброблюються одна за одною, прийдуть на один і той же вузол. Тоді час очікування пересилки між вузлами буде 0. Якщо вважати вузли рівними (з точки зору розподілу кількості заявок, які оброблюються), то ймовірність така – *1/n*. Якби всі заявки були записами, то ймовірність кешування також була б *1/n*, але, за наявності в трафіку і читань (що вимагають тільки SCUR блокування), і записів, ймовірність кешування підвищується. Необхідно відзначити, що якщо вузол володіє SCUR блокуванням на сторінку, то протягом наступних читань сторінка буде читатися з локального кешу, до того моменту, як буде витіснена звідти, згідно з вимогами алгоритму LRU (last recent used) [81],

або до моменту приходу XCUR блокування, яке інвалідує всі стани SCUR на всіх вузлах. Оскільки при моделюванні вважаємо, що обсяг пам'яті досить великий, щоб виключити витіснення сторінки відповідно до вимог LRU, то єдиною подією інвалідації, а отже, і неможливістю використовувати запитану сторінку з локального кешу, буде прихід запиту на запис (який вимагає XCUR блокування) на будь-який з вузлів системи. З огляду на вищевикладене, можна стверджувати, що потік записів разом з читаннями заповнює локальний кеш вузла, а потік тільки записів інвалідує. Якщо попередня заявка була записом, то ймовірність кешування *1/n*, а якщо читанням, то ситуація складніша, оскільки на ймовірність кешування впливає як інтенсивність читання актуальної версії сторінки на вузол, так і інтенсивність інвалідації. Розпишемо спочатку тільки залежність ймовірності кешування від інтенсивності читання актуальної версії сторінки вузлом, без урахування інвалідації за час *t*, в табл. 2.7.

Таблиця 2.7

Кількості конфліктів, та відповідні кількостям тривалості з урахуванням кешування та ймовірності

1	*2*		*k*
1-(1-1/n)	*1-(1-1/n)^2*	...	*1-(1-1/n)^k*
$(\nu_r+\nu_w)\,t\,e^{-(\nu_r+\nu_w)t}$	$\frac{\left((\nu_r+\nu_w)t\right)^2}{2!}e^{-(\nu_r+\nu_w)t}$		$\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}e^{-(\nu_r+\nu_w)t}$

У першому рядку таблиці 2.7 відображено кількість конфліктів, у другому – ймовірність того, що при відповідній кількості конфліктів сторінка виявиться закешованою, а в третьому рядку – ймовірність того, що буде відповідна кількість конфліктів. Таким чином, середня ймовірність кешування від часу без врахування інвалідації $p_{ch.n.i}$ дорівнює:

$$p_{ch.n.i}(t)=\sum_{k=1}^{\infty}\left(\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}e^{-(\nu_r+\nu_w)t}-\left(1-\frac{1}{n}\right)^k\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}e^{-(\nu_r+\nu_w)t}\right) \quad (2.12)$$

Перетворимо отриманий в 2.12 вираз:

$$p_{ch.n.i}(t)=\sum_{k=1}^{\infty}\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}e^{-(\nu_r+\nu_w)t}-\sum_{k=1}^{\infty}\left(1-\frac{1}{n}\right)^k\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}e^{-(\nu_r+\nu_w)t}=$$

$$=\left(1-e^{-(\nu_r+\nu_w)t}\right)-\sum_{k=1}^{\infty}\left(1-\frac{1}{n}\right)^k\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}e^{-(\nu_r+\nu_w)t}=$$

$$=\left(1-e^{-(\nu_r+\nu_w)t}\right)-e^{-(\nu_r+\nu_w)t}\sum_{k=1}^{\infty}\left(1-\frac{1}{n}\right)^k\frac{\left((\nu_r+\nu_w)t\right)^k}{k!}=$$

$$=\left(1-e^{-(\nu_r+\nu_w)t}\right)-e^{-(\nu_r+\nu_w)t}\sum_{k=1}^{\infty}\frac{\left(\left(1-\frac{1}{n}\right)(\nu_r+\nu_w)t\right)^k}{k!}=$$

$$=\left(1-e^{-(\nu_r+\nu_w)t}\right)-e^{-(\nu_r+\nu_w)t}\left(e^{\left(1-\frac{1}{n}\right)(\nu_r+\nu_w)t}-1\right)=$$

$$=1-e^{-(\nu_r+\nu_w)t}-\left(e^{-\frac{(\nu_r+\nu_w)}{n}t}-e^{-(\nu_r+\nu_w)t}\right)=$$

$$=1-e^{-(\nu_r+\nu_w)t}-e^{-\frac{(\nu_r+\nu_w)}{n}t}+e^{-(\nu_r+\nu_w)t}=1-e^{-\frac{(\nu_r+\nu_w)}{n}t}$$

Таким чином, отримуємо остаточну формулу:

$$p_{ch.n.i}(t)=1-e^{-\frac{(\nu_r+\nu_w)}{n}t} \quad (2.13)$$

Звернемо увагу, що якщо за час *t* не буде інвалідацій, то ймовірність кешування буде розраховуватись за формулою 2.13. А значить, якщо ми обчислимо ймовірність того, що за час *t* не буде інвалідацій і помножимо на отримане у 2.13 значення, ми одержимо залежність ймовірності кешування від часу з моменту останнього читання сторінки вузлом. Розпишемо ймовірність відсутності заявок за час *t* для певного вузла. З точки зору вузла, інвалідуючими є всі записи на інші вузли. Інтенсивність інвалідації в такому випадку дорівнює:

$$\nu_{inv} = \nu_w(1-1/n).$$

Тоді ймовірність того, що тривалість між моментом кешування і

інвалідацією буде *t*, враховуючи що потік інвалідації – пуасонівский потік, буде:

$$p_{inv}(t)=1-e^{-(1-\frac{1}{n})\nu_w t}$$

та, відповідно, густина ймовірності того, що тривалість буде *t*:

$$p_{inv}'(t)=(1-\frac{1}{n})\nu_w e^{-(1-\frac{1}{n})\nu_w t}$$

Для отримання середньої ймовірності кешування необхідно проінтегрувати добуток густини ймовірності p_{inv}' і ймовірності кешування без інвалідацій $p_{ch.n.i}$:

$$p_{cch}=\int_0^\infty (1-\frac{1}{n})\nu_w e^{-(1-\frac{1}{n})\nu_w t}\left(1-e^{-\frac{(\nu_r+\nu_w)}{n}t}\right)dt$$

Перетворюємо:

$$p_{cch}=(1-\frac{1}{n})\nu_w\int_0^\infty e^{-(1-\frac{1}{n})\nu_w t}\left(1-e^{-\frac{(\nu_r+\nu_w)}{n}t}\right)dt=$$

$$=(1-\frac{1}{n})\nu_w\int_0^\infty\left(e^{-(1-\frac{1}{n})\nu_w t}-e^{-\left(\nu_w(1-\frac{1}{n})+\frac{(\nu_r+\nu_w)}{n}\right)t}\right)dt=$$

$$=-\frac{(1-\frac{1}{n})\nu_w}{(1-\frac{1}{n})\nu_w}e^{-(1-\frac{1}{n})\nu_w t}\Bigg|_0^\infty+\frac{(1-\frac{1}{n})\nu_w}{\left((1-\frac{1}{n})\nu_w+\frac{(\nu_r+\nu_w)}{n}\right)}e^{-\left(\nu_w(1-\frac{1}{n})+\frac{(\nu_r+\nu_w)}{n}\right)t}\Bigg|_0^\infty$$

$$p_{cch}=1-\frac{(1-\frac{1}{n})\nu_w}{\left(\nu_w(1-\frac{1}{n})+\frac{(\nu_r+\nu_w)}{n}\right)}=\frac{\frac{(\nu_r+\nu_w)}{n}}{\left(\nu_w(1-\frac{1}{n})+\frac{(\nu_r+\nu_w)}{n}\right)}=\frac{(\nu_r+\nu_w)}{n\nu_w-\nu_w+\nu_r+\nu_w}$$

Звідки отримуємо формулу для коефіцієнта кешування:

$$p_{cch}=\frac{\nu_r+\nu_w}{n\nu_w+\nu_r} \quad (2.14)$$

Враховуючи, що в контексті дослідження затримок більший інтерес представляє коефіцієнт незакешування φ рівний $1\text{-}p_{cch}$, наведемо его значення:

$$\varphi = \frac{\nu_w (n-1)}{n\nu_w + \nu_r} \quad (2.15)$$

2.2.3. Обробка ресурсу в транзакції

Розглянемо послідовність дій і відповідних їм затримок при виконанні складної транзакції. Відразу відзначимо, що обробка транзакції складається з послідовної обробки ресурсів цієї транзакції. У свою чергу ресурси розташовуються на сторінках, або точніше сказати згруповані в сторінки. Сторінкова організація пам'яті в традиційних (не хмарних) СКБД мала на меті мінімізацію звернення до дискових носіїв при обробці ресурсів. Однак зворотною стороною згрупування ресурсів в сторінки стало збільшення ймовірності конфлікту за сторінки в порівнянні з імовірністю конфлікту за ресурси. При трансформації традиційних СКБД в хмарні (які використовують *shared everything* архітектуру) між традиційними рівнями доступу до сторінки (з локальної пам'яті; з диска) додався ще один: читання з віддаленої пам'яті. Оскільки час звернення до віддаленої пам'яті набагато більше часу звернення до локальної, затримки, що зумовлені процедурами вирішення конфліктів за сторінки збільшуються. На практиці це виражається в ескалації конфліктів, тобто всі затримки традиційних конфліктів за спільні ресурси різко зростають [82]. Однак в контексті нашого дослідження ключовим є те, що при виконанні транзакції час звернення до кожного ресурсу збільшується.

Розглянемо докладніше послідовність обробки ресурсу в транзакції. Спочатку необхідно отримати сторінку з ресурсом. Потім, в разі якщо ресурс не виявився заблокованим, можна перейти до обробки наступного ресурсу, але якщо ресурс виявився заблокованим – необхідно запросити

«блокування» і дочекатися його отримання. Відзначимо конструктивну особливість обробки в Oracle RAC: на самій сторінці в заголовку існує структура метаданих, яка називається «блокування» (lock) [83], що містить позначку про те, що ресурс заблокований і дані про транзакцію, яка ексклюзивно володіє ресурсом до свого завершення. Крім того, для управління конфліктами за ресурс створюється об'єкт «черга» (enqueue) [83]. Надалі в праці при розташуванні даних термінів в лапках ми будемо мати на увазі саме об'єкт типу «черга» або «блокування». У разі RAC конфігурації, «черги» до різних ресурсів розподілені між різними вузлами і перевірка заблокованості ресурсу, якщо відповідне ресурсу «блокування» присутнє на сторінці, вимагає звернення до вузла, що відстежує «чергу» до даного ресурсу. Тобто, отримавши сторінку, ми можемо знати, чи знаходиться ресурс в обробці будь-якої транзакцією, і якщо знаходиться, то адресу «черги», що керує доступом до даного ресурсу. Адреса містить ідентифікатор вузла, де розташована «черга» та її ідентифікатор. Звертаючись до майстер-вузлу «черги» сесія, яка запитує доступ до ресурсу, ставить заявку на ресурс в чергу і чекає від «черги» підтвердження можливості записати на сторінку «блокування» на запитуваний ресурс із зазначенням ідентифікатора і метаданих своєї транзакції. Зауважимо, що після завершення транзакції «черзі» відсилається повідомлення про завершення, після чого «черга» просуває чергу до ресурсу підтверджуючи наступній заявці ексклюзивне блокування. Важливим моментом є те, що після закінчення транзакції ми не оновлюємо на всіх сторінках інформацію про заблокованність ресурсів і «чергах», а робить це наступна сесія при першому, після завершення транзакції зверненні до сторінки. Це означає, що вузол, отримуючи сторінку з «блокуванням», не знає, чи активне це «блокування» – тобто транзакція, що його поставила є активною, або завершеною. Для визначення активності «блокування» і транзакції, що її встановила, здійснюється звернення до вузлу, який володіє «чергою». Той в

разі відсутності черги (тобто остання транзакція «черги» є завершена і об'єкт «черга» видалений) надсилає повідомлення, що черга порожня, і вузол, що запитав ресурс, в разі читання ресурсу знімає «блокування», а в разі запису (або блокуючого читання) ресурсу ставить своє «блокування» і створює чергу на своєму вузлі. Варто вказати, що створення і видалення об'єкта «черга» кожен раз мають на меті мінімізацію пересилань, а саме – в локальну «чергу» першій заявці швидше вносити зміни ніж в віддалену. Тобто майстром-вузлом ресурсу кожен раз призначується той вузол, що створює «чергу».

Також хотілося б відзначити, що розташування об'єкта «черга» безпосередньо в заголовку сторінки спростило б процедуру обробки ресурсу в OLTP системах, але історично склалося, що структури Oracle, які керують доступом, розташовувалися окремо від структур з даними. Рознесення даних і метаданих дає перевагу при виконанні транзакцій, що обробляють великий обсяг інформації, оскільки транзакції, що входять до «черги» працюють не з однією сторінкою.

2.2.3. Часові затримки при обробці ресурсу в транзакції

2.2.3.1. Затримки під час читання ресурсу в транзакції

Відзначимо, що будь-який трафік характеризується транзакціями з яких він складається. Визначимо поняття клас транзакції і клас ресурсу. Класом ресурсу назвемо множину ресурсів, що описують сутність [73] одного типу: наприклад рядки однієї таблиці, входи одного індексу, лоб дані, що відповідають одному атрибуту [74], тощо. Читання або зміну ресурсу назвемо маніпуляцією. Клас транзакцій визначимо як множину транзакцій, які виконують одну і ту ж послідовність маніпуляцій. Наприклад, класом є всі транзакції, які списують гроші з рахунку (береться до уваги клас ресурсу «рахунок»), після цього нараховують бонус (клас ресурсу «бонус»), а потім завершуються. З точки зору даного дослідження

важливо відзначити, що тривалості транзакцій одного класу ми будемо вважати однаковими.

Оскільки маніпуляція ресурсом може бути читанням або записом, і дії в кожному з цих випадків різні, для оцінки часу маніпуляції ресурсом необхідно розглянути послідовність дій в обох випадках. Почнемо з послідовності дій при читанні ресурсу в транзакції.

Отримавши сторінку ми перевіряємо наявність на сторінці «блокування» на цей ресурс. Якщо «блокування» існує – запитуємо «чергу», чи активна транзакція, яка це блокування утримує. Залежно від відповіді «черги» або знімаємо блокування для завершеною транзакції (і видаляємо чергу), або для активної транзакції формуємо версію сторінки «до початку цієї самої транзакції» застосовуючи дані *undo*. Таким чином, затримки при читанні ресурсу – це затримки на доступ до сторінки даного класу ресурсів, затримки на доступ до черги даного класу ресурсів і затримки на формування CR версії сторінки, а саме – читання сторінки з undo інформацією. CR – означає consistent read, тобто «До початку всіх незавершених на момент запиту транзакцій»

$$t_i = t_{acc.i} + p_{rl.i} t_{rl.i} + p_{undo.i} t_{acc.i} \quad (2.16)$$

Пояснимо часи і ймовірності, що входять в формулу 2.16: $t_{acc.i}$ – це час доступу до сторінки *i*-того класу ресурсу; $t_{rl.i}$ – це час двох пересилань: вузлу, що володіє «чергою» та відповідь від нього – пересилання ексклюзивного блокування на ресурс, що запитується; $p_{rl.i}$ – це ймовірність того, що довелося звертатися до об'єкта «черга» на іншому вузлі, оскільки в заголовку сторінки було присутнє «блокування» із зазначенням адреси черги, що відповідає їй і сама «черга» виявилася не на локальному вузлі; $p_{undo.i}$ – це ймовірність запиту *undo* даних з віддаленого вузла при читанні ресурсу *i*-того класу;

Формула для значення розглянутого часу $t_{rl.i}$:

$$t_{rl} = 2t_{net} \quad (2.17)$$

Розрахуємо ймовірність нелокального розташування черги $p_{rl.i}$. Зауважимо, що «блокування» знімається, коли «черга» при зверненні виявилася порожньою. Також зауважимо, що просуваючи чергу і наштовхуючись на її закінчення, вузлу, який отримує підтвердження ексклюзивного «блокування» про останню в черзі транзакцію, також відсилається повідомлення про те, що «черга» порожня.

Таким чином, ймовірність нелокального розташування черги $p_{rl.i}$ – це ймовірність відсутності «простою ресурсу» між двома сусідніми транзакціями. Тобто, під час транзакції, що включає обробку ресурсу прийшла ще одна або більше транзакцій (можливо різних класів), що вимагають ексклюзивного блокування даного ресурсу. Тобто, це ймовірність приходу заявки на запис-маніпуляцію за час транзакції. З урахуванням того, що наступна транзакція, що використовує розглянутий ресурс, може прийти на той же вузол що і попередня з ймовірністю

$$p_{l.noncache} = 1 - \frac{1}{n} \ ,$$

інтенсивність що визначає $p_{rl.i}$ буде:

$$\nu_{l.noncache} = \nu_{l.i}\left(1 - \frac{1}{n}\right)$$

Таким чином, якщо тривалість транзакції t_{wtl}, залежність $p_{rl.i}$ від тривалості транзакції така:

$$p_{rl.i}(t_{wtl}) = 1 - e^{-\left(1-\frac{1}{n}\right)\nu_{l.i}t_{wtl}} \quad (2.18)$$

Якщо клас ресурсу задіяний у декількох класах транзакцій, що мають різні тривалості, то інтенсивності звернення до ресурсу, відповідні різним класам транзакцій сумуються, а сумарна ймовірність – це добуток ймовірності, що відповідає часу обробки j-того класу транзакції на вагу класу (інтенсивність запитування) в сумарній інтенсивності звернення до

ресурсу. Тобто $p_{rl.i}$ залежить від інтенсивностей виникнення класів транзакцій і їх тривалостей:

$$p_{rl.i} = \sum_{j \in TRRS_i} \left(1 - e^{-\left(1-\frac{1}{n}\right)\nu_{l.i} t_{wtl.j}}\right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \quad (2.19)$$

де $TRRS_i$ – множина класів транзакцій, в які залучений i-тий ресурс, $\nu_{t.j}$ – інтенсивність j-того класу транзакції, $t_{wtl.j}$ – тривалість j-того класу транзакції. Для зручності, ресурси будемо індексувати буквою i, а транзакції – j. Тепер розпишемо ймовірність p_{undo}. Відтворювати стан потрібно коли запит на читання ресурсу i-того класу прийшов під час активної транзакції. Нагадаємо, що для внесення змін до сторінки потрібний XCUR стан, а для читання – SCUR, навіть якщо виявиться, що сторінка містить дані незафіксованих транзакцій. Отримуючи SCUR ми не знаємо, чи буде потрібно відкочувати незафіксовані зміни трансформуючи сторінку в CR (на практиці створювати додаткову CR копію або шукати в кеші версію сторінки, яка була раніше створена на необхідний момент часу). Це залежить від того, чи змінюється активною транзакцією запитуваний ресурс або ні. Таким чином, ймовірність p_{undo} – це ймовірність того, що під час активної транзакції прийшов запит на читання цього ресурсу с іншого вузла, оскільки, в разі виконання активної транзакції і читання на одному вузлі, до інших вузлів звертатися не потрібно, і час доступу буде 0, а не $t_{acc.i}$. Оскільки кількість ресурсів на сторінці – частка інтенсивності запису ν_w та інтенсивності блокування ν_l, інтенсивність читання ресурсу:

$$\nu_{r.l} = \frac{\nu_r}{\nu_w} \nu_l$$

Тоді ймовірність того, що за час тривалості транзакції t_{wtl} буде хоча б одне читання ресурсу, і воно буде не з того вузла, де знаходиться активна транзакція:

$$p_{r.l} = 1 - e^{-\left(1-\frac{1}{n}\right)\left(\frac{\nu_r}{\nu_w}\nu_l\right)t_{wtl}} \quad (2.20)$$

Аналогічно, для класу ресурсу залученого в кілька транзакцій різних класів, кожен зі своєю тривалістю, також, як була розрахована формула 2.19 з формули 2.18, отримуємо $p_{undo.i}$ – ймовірність запиту *undo* даних для ресурсу *i*-того класу, залученого в класи транзакцій з множини $TRRS_i$:

$$p_{undo.i} = \sum_{j \in TRRS_i} \left(1 - e^{-\left(1-\frac{1}{n}\right)\left(\frac{\nu_{r.i}}{\nu_{w.i}}\nu_{l.i}\right)t_{wtl.j}}\right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \quad (2.21)$$

2.2.3.2. Затримки під час запису ресурсу в транзакції

Для внесення змін у ресурс, після отримання сторінки ми перевіряємо наявність на ній «блокування» на цей ресурс. Якщо «блокування» не існує – створюємо «чергу» на поточному вузлі, і в заголовок сторінки записуємо «блокування» на оброблюваний ресурс з адресою «черги». Якщо існує «блокування» – відсилаємо «черзі» запит на обробку ресурсу і чекаємо від неї підтвердження можливості вносити зміни в ресурс, тобто підтвердження права записати в заголовок сторінки «блокування» на оброблюваний ресурс. У запиті «черзі» передається ідентифікатори транзакції і вузла, який її виконує. Отримавши «блокування» на ресурс, якщо ексклюзивне блокування на сторінку (XCUR) було віддане – вузол, затримки якого ми розглядаємо, повторно перезапитує сторінку з XCUR блокуванням. Після цього вузол вже вносить зміни в дані і в метадані сторінки – в заголовок записується інформація про «блокування» і, нарешті, сам ресурс може бути змінений. Зауважимо, що при блокуючому читанні в заголовок сторінки записується блокування на ресурс, що блокується, але сам ресурс не змінюється. Таким чином, якщо маніпуляція ресурсу *i*-того класу виявилася записом, то отримуємо формулу для часу обробки *i*-того класу ресурсу:

$$t_i = t_{acc.i} + p_{rl.i} t_{rl} + p_{inv.i} t_{acc.i} + t_{queue.i} \quad (2.22)$$

На відміну від читання-маніпуляції, час якого відображено у формулі 2.16, у нас відсутні дії по формуванню CR образу сторінки (запити undo), але присутні затримки перезапиту сторінки $p_{inv}t_{acc.i}$ та затримки очікування черги t_{queue}. $p_{inv.i}$ – це ймовірність повторного перезапиту сторінки з ресурсом *i*-того класу.

Знайдемо ймовірність повторного перезапиту сторінки p_{inv}. Сторінка перезапитується повторно, якщо з моменту першого читання сторінки з ресурсом протягом очікування запиту до «черги» і самої черги, був або *r* або *w* запит сторінки. Однак, якщо «блокування» немає, то можна відразу ж вносити зміни в сторінку, а якщо «черга», крім нашої транзакції, містить хоча б одну транзакцію, що оброблюється, та одну транзакцію, що очікує, то транзакція, яка очікує, отримавши блокування на ресурс перезапитає сторінку у вузла, затримки якого ми розглядаємо. Таким чином, для транзакції *j*-того класу, в разі нульової довжини черги ми нічого не чекаємо, p_{inv} = 0, в разі однієї транзакції в черзі – перезапит буде, якщо за час очікування черги t_{queue} прийде одна заявка, а в разі більшої кількості транзакцій в черзі p_{inv} = 1. Відобразимо це у формулі:

$$p_{inv.i}(t) = 1 - e^{-\nu_l t_{wtl.j}} - \nu_l t_{wtl.j} e^{-\nu_l t_{wtl.j}} + \nu_l t_{wtl.j} e^{-\nu_l t_{wtl.j}} \left(1 - e^{-(\nu_w + \nu_r) t_{queue.i}}\right)$$

Перетворивши отримуємо:

$$p_{inv}(t) = 1 - e^{-\nu_l t_{wtl.j}} - \nu_l t_{wtl.j} e^{-(\nu_w + \nu_r) t_{queue.i} + \nu_l t_{wtl.j}} \quad (2.23)$$

Для *i*-того ресурсу, залученого в декілька транзакцій:

$$p_{inv.i} = \sum_{j \in TRRS_i} \left(1 - e^{-\nu_{l.i} t_{wtl.j}} - \nu_{l.i} t_{wtl.j} e^{-(\nu_w + \nu_r) t_{queue.i} + \nu_{l.i} t_{wtl.j}}\right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \quad (2.24)$$

Зауважимо, що з огляду на вищеописані послідовності обробки, для одноресурсних класів транзакцій, в разі приходу однієї заявки, перезапиту не виникає. Це пов'язано з тим, що поки ми отримуємо сторінку транзакція завершується. Для цього випадку ймовірність інвалідації:

$$p_{inv}(t)=1-e^{-\nu_l t_{wtl}}-\nu_l t_{wtl} e^{-\nu_l t_{wtl}}+\nu_l t_{wtl} e^{-\nu_l t_{wtl}}\left(1-e^{-(\nu_w+\nu_r)t_{wtl}}\right) \quad (2.25)$$

Розпишемо $t_{queue.i}$. Так само, як і для черги на сторінку, потік заявок до ресурсу пуасонівський. І ймовірність додавання транзакції в чергу – це ймовірність появи заявки за час транзакції t_{wtl}. Однак, в разі якщо черга не порожня – обробка транзакції збільшується на час очікування черги, причому перша транзакція в черзі в середньому очікує половину тривалості транзакції, а наступні – всю тривалість транзакції. Відобразимо цю залежність в табл. 2.8, де в першому рядку розташуємо кількість конфліктів, у другому – тривалості, що відповідають кількостям з першого рядка, а у третьому – ймовірності відповідної кількості конфліктів.

Таблиця 2.8

Кількості конфліктів, та відповідні кількостям тривалості та ймовірності

1	*2*		*k*
$\frac{1}{2}t_{wtl}$	$\frac{3}{2}t_{wtl}$	...	$(k-\frac{1}{2})t_{wtl}$
$\nu_{l.i}t_{wtl}e^{-\nu_{l.i}t_{wtl}}$	$\frac{(\nu_{l.i}t_{wtl})^2}{2!}e^{-\nu_{l.i}t_{wtl}}$	...	$\frac{(\nu_{l.i}t_{wtl})^k}{k!}e^{-\nu_{l.i}t_{wtl}}$

Таким чином, для ресурсу *i*-того класу:

$$t_{queue.i}=\sum_{k=1}^{\infty}(k-\frac{1}{2})t_{wtl}\frac{(\nu_{l.i}t_{wtl})^k}{k!}e^{-\nu_{l.i}t_{wtl}}=$$
$$=t_{wtl}\sum_{k=1}^{\infty}k\frac{(\nu_{l.i}t_{wtl})^k}{k!}e^{-\nu_{l.i}t_{wtl}}+\frac{t_{wtl}}{2}\left(\sum_{k=1}^{\infty}\frac{(\nu_{l.i}t_{wtl})^k}{k!}e^{-\nu_{l.i}t_{wtl}}\right)$$

Перетворивши отримуємо:

$$t_{queue.i}=t_{wtl}^2\nu_{l.i}+\frac{t_{wtl}}{2}\left(1-e^{-\nu_{l.i}t_{wtl}}\right) \quad (2.25)$$

Аналогічно для ресурсу *i*-того класу, який бере участь в транзакції з множини класів $TRRS_i$:

$$p_{queue.i} = \sum_{j \in TRRS_i} \left(\nu_{rl.i} t_{wrl}^2 - \left(1 - e^{-\nu_{rl.i} t_{wrl.j}}\right) \left(\frac{t_{wrl.j}}{2} + t_{net} \right) \right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \quad (2.27)$$

2.2.4. Підсумкова формула затримок тривалості транзакції в контексті дослідження конфліктів за ресурси і сторінки

Виведені в розділах 2.2.3.1. і 2.2.3.2 формули дозволяють написати систему рівнянь. Складність полягає у взаємозалежностях t_{wtl} та t_{queue}, в першу чергу та взаємозалежностях ймовірностей $p_{rl.i}$, $p_{inv.i}$ та $p_{undo.i}$ – в другу, оскільки ймовірності в меншій мірі впливають на тривалість транзакції. Запишемо уніфіковану формулу для обробки всіх ресурсів транзакції. Будемо використовувати символи $\sigma_{w.i}$ і $\sigma_{r.i}$ для визначення того, обробка i-того ресурсу в j-тій транзакції буде читанням або записом. Для запису $\sigma_{w.i}$=1 та $\sigma_{r.i}$=0, і відповідно для читання $\sigma_{w.i}$=0 та $\sigma_{r.i}$=1.

$$t_{wtl.j} = \sum_{i \in RSTR_j} \left(t_{acc.i} + p_{rl.i} t_{rl} + \sigma_{w.i} \left(p_{inv.i} t_{acc.i} + t_{queue.i} \right) + \sigma_{r.i} p_{undo.i} t_{acc.i} \right) \quad (2.28)$$

де множина класів ресурсів j-того класу транзакції (тобто тих, які обробляються транзакціями j-того класу) позначимо $RSTR_j$.

У підсумковій системі рівнянь для нумерації класів транзакцій будемо використовувати індекс j; Множина всіх класів транзакцій позначена як TR. Множина всіх класів ресурсів – RS. Крім того, використовуються множина класів ресурсів j-того класу транзакції, позначена $RSTR_j$ та множина класів транзакцій, у які залучений i-тий клас ресурсу – $TRRS_i$. Для нумерації класів транзакцій використовується індекс j (можливо j_2, j_3, тощо), а, для нумерації класів ресурсів – індекс і (також можливе i_2, i_3 , тощо). Отже, маємо:

$$\forall j \in TR \rightarrow$$
$$t_{wtl.j} = \sum_{\forall i \in RSTR_j} \left(t_{acc.i} + p_{rl.i} t_{rl} + \sigma_{w.i} \left(p_{inv.i} t_{acc.i} + t_{queue.i} \right) + \sigma_{r.i} p_{undo.i} t_{acc.0} \right)$$
$$\forall i \in RS \rightarrow$$
$$t_{queue.i} = \sum_{\forall j \in TRRS_i} \left(\nu_{rl.i} t_{wtl.j}^2 - \left(1 - e^{-\nu_{rl.i} t_{wtl.j}}\right) \left(\frac{t_{wtl.j}}{2} + t_{net} \right) \right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}}$$
$$p_{inv.i} = \sum_{\forall j \in TRRS_i} \left(1 - e^{-\nu_{l.i} t_{wtl.j}} - \nu_{l.i} t_{wtl.j} e^{-(\nu_{w.i} + \nu_{r.i}) t_{queue.i} + \nu_{l.i} t_{wtl.j}} \right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}}$$
$$p_{rl.i} = \sum_{j \in TRRS_i} \left(1 - e^{-\left(1 - \frac{1}{n}\right) \nu_{l.i} t_{wtl.j}}\right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}}$$
$$p_{undo.i} = \sum_{\forall j \in TRRS_i} \left(1 - e^{-\left(1 - \frac{1}{n}\right) \left(\frac{\nu_{r.i}}{\nu_{w.i}} \nu_{l.i}\right) t_{wtl.j}}\right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \qquad (2.29)$$

Отриману систему рівнянь можна також як і 2.11 вирішувати ітераційно, послідовно підставляючи $t_{queue.i}$, $p_{rl.i}$, $p_{inv.i}$ та $p_{undo.i}$ для отримання $t_{wtl.j}$. На першій ітерації в цьому випадку треба взяти t_{queue}=0, $p_{rl.i}$ =0, $p_{inv.i}$=0 та p_{undo} =0. Критерієм останньої ітерації може стати перевищення сумарною довжиною черг до ресурсів $RSTR_j$ для j-тої транзакції довжини цієї транзакції. Це буде означати перевантаження пов'язане з доступом до ресурсу:

$$\forall j \in TR \rightarrow$$
$$t_{wtl.j} < \sum_{\forall i \in RSTR_j} \sigma_{w.i} t_{queue.i}$$

або перевищення чергою до сторінок будь-якого класу ресурсу, що обробляється, з урахуванням того, що інтенсивність змін до сторінок збільшується у (1+ $p_{inv.i}$) рази:

$$\forall i \in RS \rightarrow$$
$$t_{send} < \left(\nu_w \left(1 + p_{inv.i}\right) + \nu_r \right) t_{wpl} t_{send} - \frac{t_{send}}{2} \left(1 - e^{-\left(\nu_w \left(1 + p_{inv.i}\right) + \nu_r\right) \left(t_{send} + t_{queue.i}\right)}\right)$$

В такому разі, ми можемо констатувати перевантаження пов'язане з перевищенням граничної інтенсивності доступу до сторінок. Зауважимо що, *RS* – множина всіх класів ресурсу, які використовуються у трафіку.

Також можливий і варіант без перевантажень. Тоді різниця значень отриманих на останній та передостанній ітерації стане менше необхідної точності. Тут треба зауважити, що точність повинна бути на порядок (а краще два) менше ніж t_{send}.

2.3. Ймовірнісна модель або модель трафіків

Процес моделювання трафіків – це в першу чергу розбиття транзакцій трафіку, і ресурсів використовуваних в трафіку на класи. Дивлячись на формулу 2.29 стає зрозумілим, що всі тривалості, що використовуються у обчисленнях, залежать від інтенсивностей $v_{w.i}$ $v_{r.i}$ та $v_{l.i}$ до кожного класу ресурсу. Крім того, на ці тривалості впливає склад класів транзакцій. Тобто, яким чином з множини всіх класів транзакцій *TR* виділяються для кожного класу ресурсу підмножина класів транзакцій, в які залучений *i*-тий клас ресурсу – $TRRS_i$ і, відповідно, для кожного класу транзакцій, яким чином формується множина класів ресурсів $RSTR_j$, які залежать від цього *j*-того класу транзакцій.

Крім того, незважаючи на те, що вхідний потік заявок пуасонівський, потоки читань і записів сторінок кожного класу ресурсу, можуть не бути найпростішими, якщо в транзакції відбувається кілька звернень до одного ресурсу. Фактично, чи буде у другому і наступних зверненнях сторінка в локальному кеші, залежить від того, чи будуть за час між зверненнями запити на цю сторінку з інших вузлів.

Таким чином, задача знаходження ймовірності повторного читання сторінки з мережі (з одного з віддалених вузлів) аналогічна задачі

знаходження p_{inv}, не за час всієї транзакції, а за час між зверненнями до ресурсу в транзакції.

Що стосується впливу додаткового звернення на часи обробки інших транзакцій – то тут ситуація складніша. Будемо вважати, що інтенсивність змінилася на додану кількість звернень. Необхідно відзначити, що це не зовсім вірно, оскільки потік вже не є пуасонівським. Але пуасонівський потік дасть більші тривалості при такій же інтенсивності, оскільки середній час між заявками більше часу між повторними зверненнями.

Розпишемо це на прикладі. Нехай у нас ресурс змінюється в транзакції 2 рази. Відобразимо в табл. 2.9 ймовірності та відповідні їм тривалості кількості заявок за час t_{wpl}. У першому рядку відображено кількість конфліктів, у другому тривалість обробки такої кількості, у третьому ймовірність, якщо потік пуасонівський з інтенсивністю доступу до ресурсу v_{pl}, у четвертому потік транзакцій пуасонівський з інтенсивністю v_{pl}, але ресурс оброблюється двічі на транзакцію, а у п'ятому – пуасонівський потік з інтенсивністю доступу до ресурсу $2v_{pl}$.

Таблиця 2.9

Затримки і ймовірності для випадку двократного звернення до ресурсу у транзакції для 3х потоків.

1	*2*		*2n-1*	*2n*
$t_{send}-\frac{t_{send}}{2}$	$2t_{send}-\frac{t_{send}}{2}$	...	$(2n-1)t_{send}-\frac{t_{send}}{2}$	$2nt_{send}-\frac{t_{send}}{2}$
$(\nu_{pl}t_{wpl})e^{-\nu_{pl}t_{wpl}}$	$\frac{(\nu_{pl}t_{wpl})^2}{2!}e^{-\nu_{pl}t_{wpl}}$	...	$\frac{(\nu_{pl}t_{wpl})^{2n-1}}{(2n-1)!}e^{-\nu_{pl}t_{wpl}}$	$\frac{(\nu_{pl}t_{wpl})^{2n}}{(2n-1)!}e^{-\nu_{pl}t_{wpl}}$
$q_{inv}(\nu_{pl}t_{wpl})e^{-\nu_{pl}t_{wpl}}$	$(1-q_{inv})(\nu_{pl}t_{wpl})e^{-\nu_{pl}t_{wpl}}$	...	$q_{inv}\frac{(\nu_{pl}t_{wpl})^{n}}{n!}e^{-\nu_{pl}t_{wpl}}$	$(1-q_{inv})\frac{(\nu_{pl}t_{wpl})^{n}}{n!}e^{-\nu_{pl}t_{wpl}}$
$(2\nu_{pl}t_{wpl})e^{-2\nu_{pl}t_{wpl}}$	$\frac{(2\nu_{pl}t_{wpl})^2}{2!}e^{-2\nu_{pl}t_{wpl}}$	...	$\frac{(2\nu_{pl}t_{wpl})^{2n-1}}{(2n-1)!}e^{-2\nu_{pl}t_{w}}$	$\frac{(2\nu_{pl}t_{wpl})^{2n-1}}{(2n-1)!}e^{-2\nu_{pl}t_{wpl}}$

де q_{inv} – ймовірність відсутності повторного читання сторінки при повторній обробці в транзакції. Розпишемо для четвертого рядка для кожного n:

$$q_{inv}\frac{\left(\nu_{pl}t_{wpl}\right)^n}{n!}e^{-\nu_{pl}t_{wpl}}\left(2n-1\right)t_{send}-\frac{t_{send}}{2}+\left(1-q_{inv}\right)\frac{\left(\nu_{pl}t_{wpl}\right)^n}{n!}e^{-\nu_{pl}t_{wpl}}2nt_{send}-\frac{t_{send}}{2}=$$
$$=t_{send}\frac{\nu_{pl}t_{wpl}^n}{n!}e^{-\nu_{pl}t_{wpl}}\left(2nq_{inv}-q_{inv}+2n-2nq_{inv}\right)-t_{send}=t_{send}\left(2n-q_{inv}-1\right)\frac{\nu_{pl}t_{wpl}^n}{n!}e^{-\nu_{pl}t_{wpl}}$$

Тоді середня тривалість черги до сторінки для цього випадку:

$$t_{p.queue}=\sum_{n=1}^{\infty}\left(2n-q_{inv}-1\right)\frac{\nu_{pl}t_{wpl}^n}{n!}e^{-\nu_{pl}t_{wpl}}t_{send}=2\nu_{pl}t_{wpl}t_{send}-t_{send}\left(1+q_{inv}\right)\left(1-e^{-\nu_{pl}t_{wpl}}\right) \quad (2.30)$$

Відзначимо, що q_{inv} залежить від ν_{pl} і співвідношення тривалості транзакції і тривалості між зверненнями всередині транзакції.

Порівнюючи з формулою 2.9, отримуємо нерівність:

$$2\nu_{pl}t_{wpl}t_{send}-t_{send}\left(1+q_{inv}\right)\left(1-e^{-\nu_{pl}t_{wpl}}\right)>\nu_{pl}t_{wpl}t_{send}-\frac{t_{send}}{2}\left(1-e^{-\nu_{pl}t_{wpl}}\right)$$

перетворюємо:

$$\nu_{pl}t_{wpl}>\left(\frac{1}{2}+q_{inv}\right)\left(1-e^{-\nu_{pl}t_{wpl}}\right) \quad (2.31)$$

що справедливо, оскільки обидві функції від добутку $\nu_{pl}\ t_{wpl}$ монотонні, в 0 обидві мають значення рівне 0, і похідна лінійної функції в діапазоні $(0,\infty)$ більше.

С іншого боку порівнявши з сумою по четвертому рядку маємо нерівність:

$$2\nu_{pl}t_{wpl}t_{send}-t_{send}\left(1+q_{inv}\right)\left(1-e^{-\nu_{pl}t_{wpl}}\right)<2\nu_{pl}t_{wpl}t_{send}-\frac{t_{send}}{2}\left(1-e^{-2\nu_{pl}t_{wpl}}\right)$$

Перетворивши яку маємо:

$$2\left(1+q_{inv}\right)\left(1-e^{-\nu_{pl}t_{wpl}}\right)>\left(1-e^{-2\nu_{pl}t_{wpl}}\right)$$

враховуючи, що

$$2\left(1+q_{inv}\right)\left(1-e^{-\nu_{pl}t_{wpl}}\right)>2\left(1-e^{-\nu_{pl}t_{wpl}}\right) \quad (2.32)$$

можемо розписати:

$$2\left(1-e^{-\nu_{pl}t_{wpl}}\right)>\left(1-e^{-2\nu_{pl}t_{wpl}}\right)$$

Обидві функції від добутку $v_{pl}t_{wpl}$ монотонні, та в 0 обидві мають значення рівне 0. Порівнявши похідні цих функцій, можемо сказати, яка з них більше:

$$2\nu_{pl}t_{wpl}e^{-\nu_{pl}t_{wpl}} > 2\nu_{pl}t_{wpl}e^{-2\nu_{pl}t_{wpl}}$$

Нерівність похідних справедлива, а значить і нерівність 2.32 справедлива. Таким чином, для класу транзакції, яка використовує двократне звернення до одного ресурсу можна розглядати внесок інтенсивності в загальну інтенсивність до класу *i*-того ресурсу в діапазоні (ν_{pl}, $2\nu_{pl}$) і для запису і для читання. Тобто, маємо мінімальну і максимальну інтенсивність для підставлення в отримані формули. Для обчислення більш точних значень черги для класів ресурсів, що задіяні в класах транзакцій з двократним зверненням до ресурсів цих класів, можна застосовувати формулу 2.30. При наявності в трафіку класів транзакцій, які звертаються до ресурсу з кратністю понад 3 можна застосовувати підхід, схожий на застосований при отриманні формули 2.30, однак в нашому дослідженні такі трафіки не розглядаються.

Крім того, відзначимо, що в контексті дослідження черг до ресурсів, інтенсивність звернення до ресурсу (блокування ресурсу) не залежить від того, скільки разів у транзакції обробляється ресурс, оскільки блокується він лише один раз.

Зауважимо, що в багатьох випадках середній інтервал між запитами на сторінку з однієї транзакції на порядок менше ніж інтервал між транзакціями. Це має наступні наслідки.

- Якщо з двох запитів на обробку сторінки читання відбувається першим, а запис – другим, то інтенсивністю читання можна знехтувати. Це пов'язано з тим, що надбавка в потік читання, що здійснює кешування, (що було розглянуто в розділі 2.2.2) мінімальна, тобто, закешованою сторінкою, швидше за все, скористається тільки маніпуляція запису цієї ж транзакції, наступна за читанням, яка в

свою чергу відразу ж інвалідує всі сторінки SCUR стану. Тоді допустимим є ігнорування інтенсивності, що додається маніпуляцією-читанням перед маніпуляцією-записом.

- Якщо ж маніпуляція-запис відбувається перед читанням – то надбавку інтенсивності запису ігнорувати не можна оскільки інвалідуючий потік в коефіцієнті кешування (2.16) в n раз більш значущий.
- При послідовній (не перемежованій зверненням до інших ресурсів) обробці саме одного і того ж ресурсу (а не ресурсів класу) два читання має сенс розглядати як одне читання, два записи – як один запис, а читання і запис – як запис.

При розрахунку трафіків в розділі 4 буде використовуватися спрощений підхід, що оцінює мінімальні і максимальні інтенсивності на основі вищенаведених фактів.

2.4. Висновки по розділу 2

1. На основі проведеного аналізу запропонована трикомпонентна модульна математична модель, що дозволяє визначити сумарні затримки процедур доступу до сторінки і процедур, що забезпечують коректність обробки загального ресурсу в Oracle RAC.
2. В рамках запропонованої моделі розглянуті і формалізовані сценарії обробки сторінки в Oracle RAC. Визначено чинники, що визначають ймовірності сценаріїв.
3. Досліджено вплив кешування на швидкість доступу до сторінок. Отримана формула для оцінки ймовірності кешування.
4. Досліджено конфлікт за сторінки пам'яті. Запропоновано систему рівнянь і обраний чисельний метод для знаходження рішення для знаходження довжини черги до сторінок класу ресурсів.
5. Досліджено затримки багаторесурсної транзакції, виявлені причини,

що збільшують тривалість транзакції при збільшенні інтенсивності транзакцій. Оцінені ймовірності цих подій і тимчасові затримки, пов'язані з ними.

6. Визначено і відображена в системі рівнянь взаємна залежність тривалостей транзакцій і тривалостей черг до ресурсів, що входять в ці транзакції. Обрано чисельний метод для вирішення системи рівнянь.
7. Обрана методика формалізації трафіку, обґрунтовано використання пуасонівських потоків при моделюванні, визначений інтерфейс взаємодії моделі трафіків з моделлю конфліктів.

РОЗДІЛ 3. Розробка способу організації одночасного доступу в розподілених БД

3.1. Методи удосконалення способу обробки сторінки

Для вирішення проблем описаних в пп. 1.5.1 та 1.5.2 пропонуються три методи. Метод двофазної обробки транзакції, що дозволяє виключити блокування ресурсу на етапі отримання списку ресурсів, і таким чином істотно скоротити час блокування. Метод конвеєризації фаз, який модифікує двофазну обробку, дозволяючи в ситуації низького відсотка надмірності даних на вузлах зберегти ефективність методу двофазної обробки даних. І метод призначення обробника спільного ресурсу, який дає можливість розвантажити чергу до розподілених сторінок пам'яті в разі збільшення цієї черги при перевищенні граничної інтенсивності та дозволяє залишити систему в робочому стані навіть при перевищенні інтенсивностями звернення до критичних ресурсів граничних для стандартного способу доступу значень. Всі три запропонованих методи дозволяють вирішити поставлену задачу оптимізації алгоритмів *shared everything* архітектури для роботи в глобальних мережах.

3.1.1 Метод двофазної обробки транзакції.

У складній транзакції, що була розглянута у розділі 1.5.2, обробка кожного ресурсу послідовна. Це означає, що ми послідовно ставимо SCUR або XCUR блокування на сторінки всіх ресурсів, що обробляються в складній транзакції. Зрозуміло, що при виконанні транзакції послідовність дій має значення, однак ідеальним варіантом був би такий, коли всі необхідні для виконання транзакції сторінки і ресурси (рядки таблиць, входи індексів і т.д.) надавалися для виконання транзакції з усіх вузлів до початку транзакції. Пропонований метод двофазної обробки транзакції дозволяє виключити послідовне блокування сторінок і ресурсів на етапі їх

отримання, і, як наслідок – скоротити затримки транзакцій за рахунок асинхронного отримання ресурсів замість послідовного.

Під час першої фази ми виконуємо транзакцію використовуючи всі наявні на вузлі закешовані (в т.ч. неактуальні CR – consistent read) сторінки, для того, щоб отримати список необхідних спільних ресурсів. У момент виконання відбувається асинхронне відправлення майстер-вузлу запиту на відповідний ресурс. За результатом першої фази ми маємо виконану транзакцію, проте можливо, що через неактуальність використаних ресурсів, транзакція вимагатиме коректувального закінчення.

Під час другої фази ми чекаємо актуальних (current) версій ресурсів (сторінок), звіряємо чи список актуальних (current) ресурсів відповідає своїм версіям використаним під час першої фази в контексті зроблених змін. При цьому актуальна версія може бути залишена як в SCUR стані (наприклад для проміжних сторінок індексу), так і в consistent read стані, для використання в інших транзакціях на цьому вузлі. Далі, якщо зміни в актуальних сторінках не вплинули на результат – то ми залишаємо отриманий на першій фазі результат і транзакція готова до фіксації (commit), якщо ж зміни істотні, то транзакція виконується заново. У деяких випадках може формуватися коректувальне закінчення транзакції. Вибір між коректуванням першої фази і відкочуванням з повторним виконанням має сенс розглянути у контексті фіксації транзакції, що виходить за рамки цієї наукової праці. На рис. 1 приведена діаграма затримок UML [86], яка показує різницю затримок підходу, що використовується у *shared everything* реалізації Oracle RAC та запропонованої процедури розв'язання конфлікту, яка використовує метод двофазної обробки транзакції. Під абревіатурою ПРК розуміється процедура розв'язання конфлікту.

Таким чином у разі успіху (актуальні (current) версії ресурсів не вплинули на результат транзакції, яка використала неактуальні (consistent read) версії) час блокування ресурсів завершується з другою фазою. У разі

неуспіху воно триває на час відсилання і підтвердження коректувального закінчення транзакції.

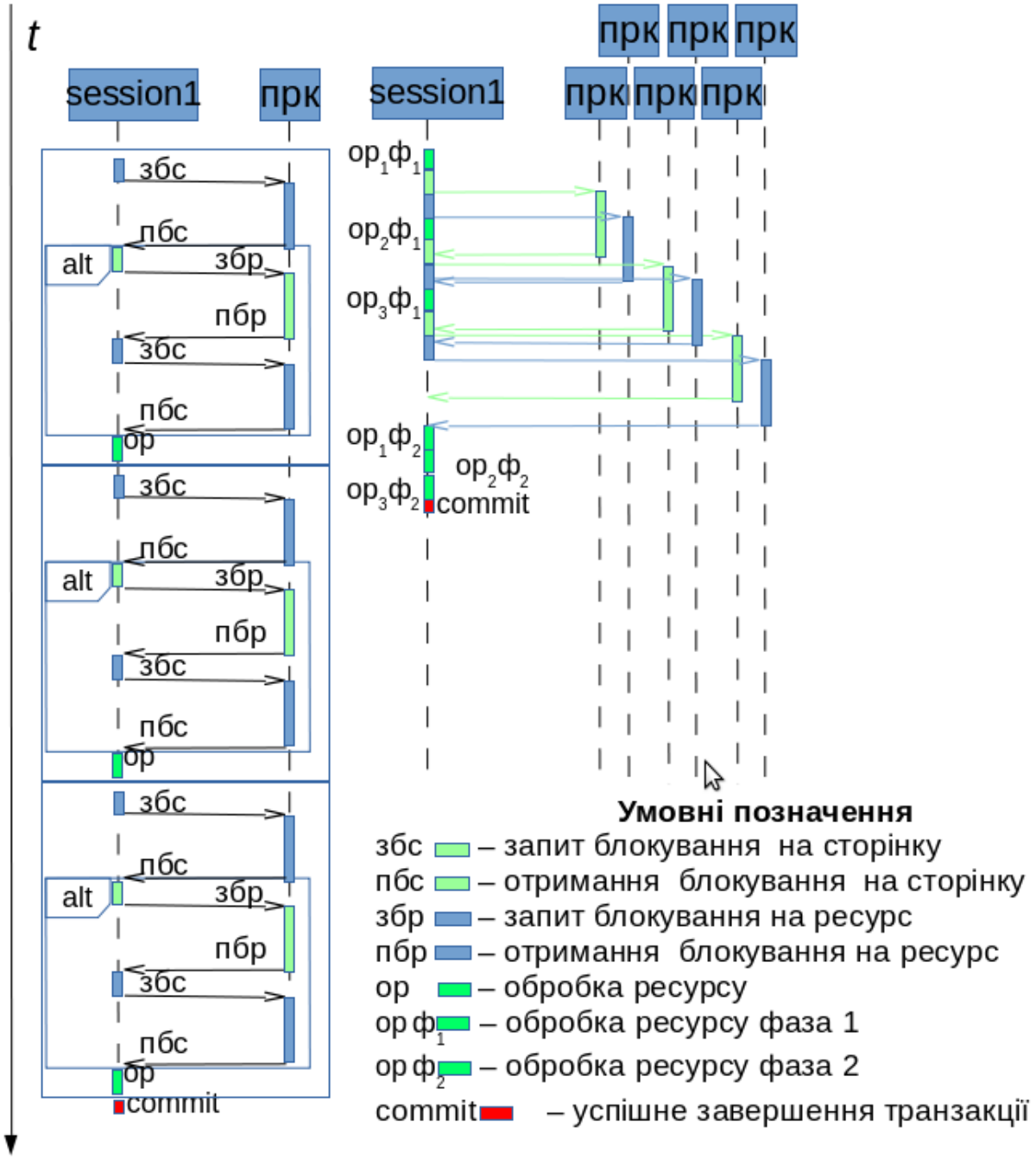

Рис. 1. Порівняння традиційної процедури розв'язання конфліктів та запропонованої, що використовує метод двофазної обробки транзакції

Зауважимо, що сама транзакція триває завжди до підтвердження коректувального закінчення або підтвердження того, що коректувального

закінчення в транзакції немає, але блокування на ресурс у разі успіху можна зняти раніше, що підвищує стійкість системи до перевантажень при обробці спільного ресурсу. Резюмуючи, можна виділити ключові особливості запропонованого методу:

- Не зважаючи на те, що обробка транзакції насправді не розпаралелюється, а потоки, що оброблюють кожен ресурс виконуються послідовно в асинхронному режимі (callback), можна вважати такий спосіб розпаралелюванням, якщо час виконання першої фази істотно менше часу пересилання між вузлами, що практично завжди справедливо для глобальних мереж.
- Розпаралелювання складної транзакції в системах, що використовують *shared everything* архітектуру, призводить часові витрати на обробку всієї транзакції до часу, порівняного з обробкою однієї сторінки.

Оцінюючи можливість практичної реалізації методу, варто вказати на спосіб контролю над виконанням транзакції в сесіях бази даних. Якщо транзакції реалізовані в збережених процедурах СКБД [87] і користувач працює з базою, викликаючи для кожної бізнес-операції свою збережену процедуру – то це ідеальний варіант для використання методу. Критичні з точки зору тривалості виконання транзакції процедури можуть бути поміченими (в т.ч. за межами свого коду), як такі, що вимагають двофазного виконання. Причому, що важливо, критичність можна відстежувати динамічно, оцінюючи динаміку зростання черг до ресурсів транзакцій.

У разі ж, якщо порядок виконання операцій і об'єднання їх в транзакції лежить за межами СКБД – то реалізація методу істотно ускладнюється. Необхідно змінювати структуру обміну повідомлень між вузлом, що виконує код клієнта, і сервером бази даних, що фактично виливається в написання нових драйверів до існуючих систем, залучених у

виконання клієнтського коду. На щастя, інкапсуляція критичних з точки зору часу виконання ділянок коду є загальноприйнятою практикою [88], і часто закладається в архітектуру додатків. Це дозволяє позитивно оцінити можливості застосування методу.

3.1.2. Метод призначення обробника ресурсу

Варто зазначити, що виникнення черги дає перевагу того, що обробку заявок в черзі можна оптимізувати для мінімізації часів пересилань між вузлами, ґрунтуючись на знанні взаємозв'язків ресурсів всіх транзакцій, що входять в чергу і їх сторінок, а також послідовностей обробки цих ресурсів. Очевидно, що найкращий варіант вийде, якщо всі транзакції пов'язані в чергу виконувати на одному вузлі. Але тоді ми впираємось в обмеження масштабованості, описані в розділі 1. Відповідно, завдання стоїть розподіляти між вузлами, які входять у хмарну систему, всі наявні у всіх чергах транзакції і послідовності обробки сторінок, за умови, що цільової функцією якості розміщення є зниження часу пересилань між вузлами. Оскільки це завдання доводиться вирішувати в реальному часі, необхідно використання простих і ефективних алгоритмів планування у процедурах розв'язання конфлікту та доступу до сторінок. Пропонована процедура доступу до сторінок, яка ґрунтується на методі призначення обробника ресурсу, задовольняє таким вимогам.

Результатом пропонованої процедури є «розвантаження», тобто зменшення до нуля довжини черги до розподіленої сторінки пам'яті, що виникла в результаті того, що інтенсивність доступу до «гарячих» ресурсів, що розташовані на цій сторінці, короткочасно або довготривало перевищила граничну інтенсивність переходу в стан «перевантаження», розглянутого в розділі 1.5.1. Ідея пропонованого методу, що лежить в основі процедури доступу, досить проста: якщо наростає черга – то необхідно знизити кількість пересилань між вузлами. Для цього

вибирається вузол, який ексклюзивно буде вносити зміни в «гарячу» сторінку. Це дозволяє уникнути черг, які утворюються через збільшення часу відклику внаслідок додавання до часу обробки тривалостей пересилань між вузлами. Важливо також закласти у процедуру можливість мігрувати з вузла, який вносить зміни, на інший вузол, для забезпечення динамічного балансування навантаження в разі кількох гарячих ресурсів транзакції. Розпишемо структуру обміну повідомленнями пропонованої процедури.

1. При виникненні довгої черги майстер-вузол повинен перевести сторінку в режим «розвантаження», тобто вузол, що виконує обробку, в певний момент часу повинен бути повідомлений про переведення сторінки в режим «розвантаження» і призначений вузлом, який це розвантаження виконує (надалі вузол-хостер). Це висуває вимоги до майстер-вузлу відстежувати історію повідомлень, що пересилаються. В іншому випадку черга почне розвантажуватися тільки після обслуговування запитів сторінок, які прийшли після моменту переведення в режим «розвантаження».
2. На запити сторінок майстер-вузол дає відповідь, із зазначенням вузла-хостера, якому треба пересилати пакет з діями, що треба виконати над сторінкою.
3. Вузол, що виконує обробку, формує і пересилає хостеру своє завдання.
4. Хостер вносить зміни в сторінку і пересилає поточну версію сторінки вузлу, що мав оброблювати її. На цьому етапі можлива оптимізація, яку ми при моделюванні враховувати не будемо: замість пересилання всієї сторінки пересилаються зміни з останнього кешування на вузлі, що мав оброблювати сторінку.
5. Вузол, що мав оброблювати сторінку, отримує її в стані «як ніби він сам вніс в неї зміни і відразу ж відправив далі за запитами інших

сторінок». Визначаючи термінологію, домовимося говорити, що сторінка знаходиться в стані «розвантаження», якщо до неї почала застосовуватись наведена процедура доступу. Виходом з «розвантаження» домовимося називати процес, за результатами якого припиняється застосування пропонованої процедури доступу, і замість нього знову починає застосовуватися традиційна процедура доступу до сторінки, притаманна *shared everything* архітектурі.

Важливим моментом в цьому алгоритмі є мінімізація ресурсів на моніторинг виникнення черги. У даній науковій праці ми відзначимо важливість цих питань, але при моделюванні будемо вважати, що черга почне розвантажуватися тільки починаючи з обслуговування запитів сторінок, які прийшли після моменту переведення в стан «розвантаження». Крім того, вихід зі стану «розвантаження», оцінка доцільності виходу і пріоритизація призначення того чи іншого вузла хостером – тобто все, що відноситься до перехідних процесів, пов'язаних з «розвантаженням», також виходить за рамки даного дослідження. UML діаграма, наведена на рис. 2, порівнює обробку декількох сторінок в *shared everything* архітектурі, як за допомогою традиційної процедури доступу, так і за допомогою процедури, що використовує метод призначення обробника ресурсу.

Резюмуючи опис методу, потрібно відзначити, що ефект виходить за рахунок того, що хоча і збільшується кількість пересилань між вузлами – але число послідовних пересилань скорочується. Застосування методу може стати ключовим при вирішенні задачі доступу до розподілених сторінок ресурсу в хмарних системах, оскільки повністю знімаються обмеження на масштабованість, пов'язані з мінімальними вимогами до каналів зв'язку між вузлами для уникання перевантажень, пов'язаних з доступом до сторінок.

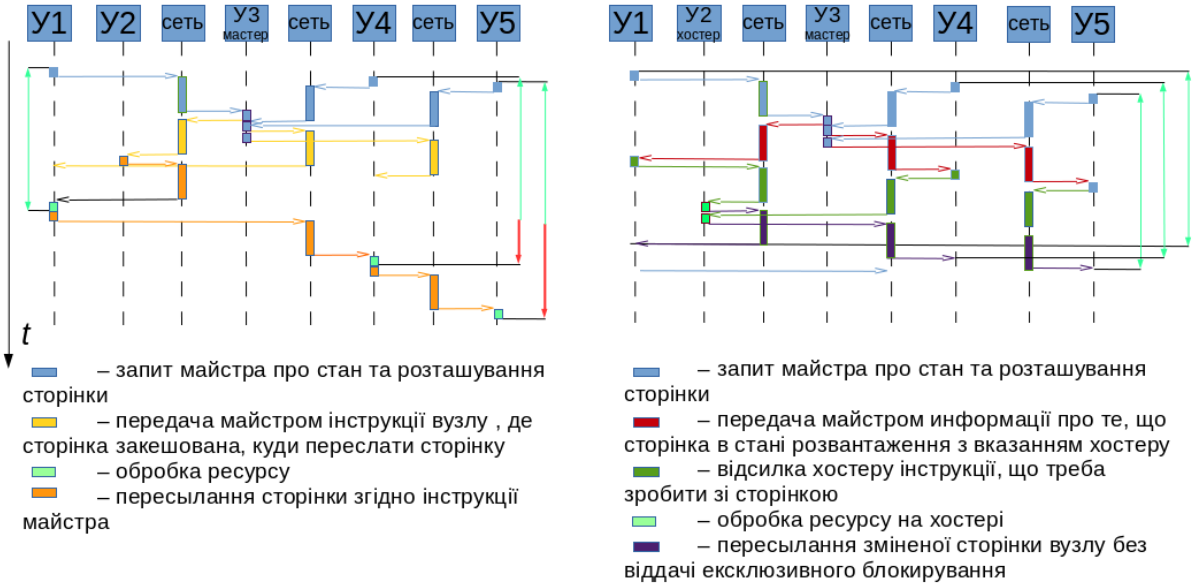

Рис. 2. Порівняння процедур доступу при обробці декількох сторінок в *shared everything* архітектурі

3.1.3. Проблема відсутності на вузлі сторінки у CR стані будь-якої версії

Наведеному методу двофазної обробки транзакції властиві деякі недоліки, що значно знижують його ефективність. Основним недоліком є відсутність будь-якої версії сторінки на вузлі, що виконує обробку. Тобто, якщо при застосуванні метода двофазної обробки, версії сторінки, що необхідна для виконання транзакції, в станах SCUR, XCUR або CR, не знайдено, то при виконанні першої фази необхідно дочекатися отримання відсутньої сторінки і тільки після цього продовжити транзакцію. Це збільшує час виконання транзакції на кількість пересилань кратну кількості подій відсутності будь-якої версії сторінки протягом виконання транзакції.

3.1.4. Метод конвеєризації фаз

Метод конвеєризації фаз є оптимізацією методу двофазної обробки транзакції при відсутності на вузлі сторінки будь-якої версії (XCUR, SCUR або CR). Суть методу конвеєризації фаз полягає в тому, що в разі розбиття послідовності транзакцій на групи, які визначаються відсутністю в

локальному кеші CR версій сторінки, друга фаза для кожної попередньої групи може виконуватися паралельно з першою фазою наступної. Такий підхід дозволяє в разі помилки, закладеної в оптимістичний підхід, провести коригувальні дії зі сторінками, ресурси яких змінилися порівняно з використаними на першій фазі, одночасно з виконанням першої фази наступної групи. Це дозволяє знизити середній час обробки транзакції. Варто відзначити, що в разі, якщо послідовність транзакцій не розпадається на групи – то конвеєризації фаз немає. На рис. 3 представлена UML діаграма, що демонструє ефективність застосування конвеєризації фаз. Зліва направо: перша діаграма демонструє ситуацію наявності всіх сторінок (можливо, що і неактуальних) в локальному кеші, діаграма в центрі – ситуацію відсутності сторінки третього ресурсу в кеші, а діаграма справа – конвеєризацію фаз, при відсутності третього ресурсу в кеші. З діаграм видно, що ефект відсутності сторінки в кеші мінімізований і час наближається до ідеального, проілюстрованого першим випадком (на діаграмі крайній зліва).

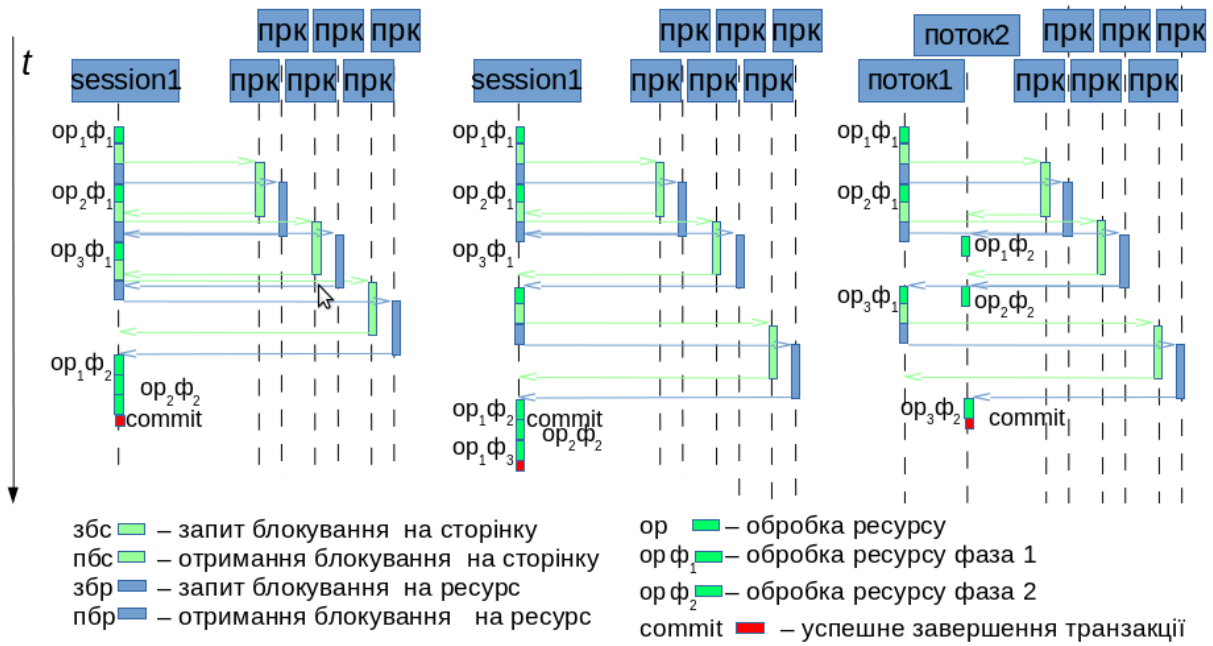

Рис. 3. Ефективність застосування конвеєризації фаз

3.2. Зміни в математичній моделі, для оцінки затримок запропонованих процедур розв'язання конфлікту

Асинхронний підхід при двофазній обробці транзакції фактично дозволяє розпаралелювати обмін пересиланнями при доступі до ресурсів і сторінок. Зауважимо, що в реальності з малим тимчасовим лагом при обробці кожного наступного ресурсу, проте тривалістю обробки в порівнянні з часом пересилання ми нехтуємо, а значить можна говорити про розпаралелювання. Крім того, в разі виникнення черг до сторінок ресурсів або до ресурсів, очікування доступу в цих чергах також виконується одночасно – тобто розпаралелюється. З огляду на все це, для оцінки часових затримок необхідно оцінити довжини черг до ресурсів і сторінок при двофазної обробці, а також ймовірності $p_{rl.i}$, $p_{inv.i}$ та p_{undo} і вплив цих ймовірностей при паралельному підході, що використовується в методі.

3.2.1. Знаходження середньої тривалості черги до сторінки якщо використовується двофазна обробка транзакції.

Оскільки ми вважаємо, що очікування як сторінки так і ресурсу виконуються паралельно, то час очікування черги при такому підході, на відміну від класичного підходу, де тривалості підзадач підсумовуються, це очікування максимальної за тривалістю черги, серед паралельно виконуваних очікувань. Це вносить суттєві зміни в алгоритм розрахунку тривалості очікування сторінок в транзакції. Зупинимося на цьому докладніше.

Нехай в транзакції бере участь i ресурсів. Літерою k позначимо кількість можливих конфліктів за сторінку одного з ресурсів. За умови, що потік звернень до сторінки пуасонівський, наведемо значення ймовірності виникнення k конфліктів за сторінки ресурсу i-ого класу від часу t:

$$p_{ik}(t) = \frac{(\varphi_i \nu_{pl.i} t)^k}{k!} e^{-\nu_{pl.i} t} \quad (3.1)$$

де $v_{pl.i}$ – інтенсивність конфліктів за сторінку i-того класу ресурсів, φ_i – коефіцієнт незакешування, розрахований за формулою 2.15: Імовірність того, що конфліктів за сторінки ресурсу i-ого класу при обробці буде не більше ніж k, буде сумою від 0 до k ймовірностей p_{ik}, розрахованих за формулою 3.1. При асинхронному (в даному випадку вважаємо паралельному) підході ймовірність не більше ніж k конфліктів для кожного з ресурсів у всіх паралельних потоках – це добуток ймовірностей «не більше ніж k конфліктів» в кожному з розпаралелених потоків. У той же час, ймовірність точно k конфліктів – це різниця ймовірності «не більше ніж k конфліктів» та ймовірності «не більше ніж (k-1) конфліктів».

Запишемо цей факт у вигляді формули, тобто залежність ймовірності виникнення k_{agg} конфліктів (позначимо $p_{k_{agg}}$) при асинхронному виконанні за час t:

$$p_{k_{agg}}(t)=\prod_{i=1}^{i_{total}}\left(\sum_{k=0}^{k_{agg}} p_{ik}(t)\right)-\prod_{i=1}^{i_{total}}\left(\sum_{k=0}^{k_{agg}-1} p_{ik}(t)\right) \quad (3.2)$$

де $p_{ik}(t)$ – ймовірність k конфліктів за сторінку i-го класу ресурсу, i_{total} – кількість ресурсів в транзакції. Відповідні ймовірності $p_{ik}(t)$ для сторінок i-того класу ресурсу обчислюються за формулою (3.1). Зауважимо, що для k_{agg} =0 формула (3.2) не визначена, оскільки від'ємник, а це сума від 0 до -1 – неможливий. Наведемо формулу для k_{agg} =0:

$$p_0(t)=\prod_{i=1}^{i_{total}} p_{i0} \quad (3.3)$$

Для наступних значень натурального числа k_{agg} формула 3.2 вже може застосовуватись. Наприклад, для k_{agg} =1 значення буде:

$$p_1(t)=\prod_{i=1}^{i_{total}}\left(\sum_{k=0}^{1} p_{ik}\right)-\prod_{i=1}^{i_{total}} p_{i0}$$

Розглянемо в такому випадку, чому дорівнює середня тривалість очікування сторінок в транзакції. Аналогічно підходу, який використовувався при виведенні формули тривалості очікування черги до

сторінки в традиційному *shared everything* (2.9), відзначимо факт, що перша прийшла заявка не обов'язково чекає повний час t_{send}, а в середньому – половину – тобто $t_{send}/2$, а решта чекають t_{send}. Застосувавши для знаходження кількості конфліктів формули 3.2 і 3.3 отримуємо математичне очікування тривалості їх обробки за час t_{wpl}, тобто середню довжину тривалості очікування сторінок в транзакції:

$$t_{p.queue} = \sum_{k=1}^{\infty} k\, p_k(t_{wpl}) - \frac{t_{send}}{2}\left(1 - p_0(t_{wpl})\right) \quad (3.4)$$

Індекс k_{agg} ми замінили більш звичним k. Також як і для розрахунку традиційної *shared everything* черги до сторінки (2.8): t_{wpl} – це час, протягом якого проводиться спостереження, $t_{p.queue}$ – середній час додаткового очікування сторінок в транзакції, викликаний конфліктом за сторінки. Оскільки це паралельне очікування черг, то в позначенні будемо також застосовувати слово queue (черга), а *p* означає page (сторінка). Оскільки:

$$t_{wpl} = t_{send} + t_{p.queue}$$

маємо систему рівнянь, аналогічну 2.11:

$$\begin{cases} t_{p.queue} = \sum_{k=1}^{\infty} k\, p_k(t_{wpl}) - \frac{t_{send}}{2}\left(1 - p_0(t_{wpl})\right) \\ t_{wpl} = t_{send} + t_{p.queue} \end{cases} \quad (3.5)$$

Також як і при традиційному *shared everything* (2.11), найбільш простий з точки зору обчислення варіант – це ітераційний. Значення в кожній наступній ітерації будуть наближатися до значень рішення системи рівнянь виходячи з збіжності послідовностей $t_{p.queue}$ та t_{wpl}, яка, в свою чергу, випливає з того, що обидві функції тривалості $t_{p.queue}$ та t_{wpl} монотонно зростають при зростанні значень своїх атрибутів t_{wpl} та $t_{p.queue}$ відповідно, а тривалості t_{wpl} та $t_{p.queue}$ – скінченні. Зауважимо, що умовою того, що тривалості t_{wpl} та $t_{p.queue}$ спрямовуються в нескінченність є перевищення на черговій ітерації значенням $t_{p.queue}$ значення t_{send}, тобто вхід у стан перевантаження, що було розглянуте у розділі 1.5.1.

3.2.2. Знаходження середньої довжини черги до ресурсу при використанні двофазної обробки транзакції

Складніше виглядає ситуація з тривалістю очікування ресурсів в транзакції, або перефразовуючи – з паралельної чергою до ресурсу. На відміну від паралельної черги до сторінки, описаної в розділі 3.4.1 час одного очікування ресурсу для різних ресурсів різний. В такому випадку має сенс використовувати *послідовність можливих тривалостей можливої черги*. Під *можливою тривалістю* тут ми розуміємо впорядковану за тривалістю можливу дискретну тривалість очікування черги. Пояснимо на прикладі: Нехай у розглянутій транзакції очікування першого ресурсу – *2t*, другого – *5t*, третього – *7t*. Тоді перша тривалість – *2t* (очікування першої транзакції 1 раз), друга – *4t* (очікування першої транзакції 2 рази), третя – *5t* (другої транзакції – 1 раз), четверта – *6t* (першої транзакції – 3 рази), п'ята – *7t* (третьої транзакції – 1 раз) і т. д. Звернемо увагу на те, що номер можливої тривалості – це номер в упорядкованому списку цих довжин за зростанням. Для нумерації за зростанням будемо використовувати індекс *l*. Якщо ресурси обробляються в різних транзакціях, то і тривалості очікування черги кратні тривалостям цих транзакцій, а значить, можливий час очікування – це сума зважених можливих очікувань до транзакцій, в яких задіяні ресурси.

Виділивши можливі тривалості очікування і упорядкувавши їх за зростанням, позначимо символом $\Omega_l(t)$ ймовірність неперевищення за час t l-ої за зростанням «можливої тривалості». Знайдемо, чому дорівнює ця ймовірність. Для обчислення значень Ω_i необхідно застосувати наступний алгоритм:

1. Обчислюється добуток ймовірностей відсутності конфліктів. Ймовірність $p_{l.i0}$, для ресурсу i-того класу, оскільки потік звертань до ресурсів пуасонівський с інтенсивністю $\nu_{l.i}$:

$$p_{l.i0} = e^{-(1-\frac{1}{n})\nu_{l.i}t_{wtl}} \quad (3.6)$$

де $t_{wtl.j}$ – тривалість транзакції в якій використовується i-тий ресурс.

Для ресурсу, залученого в декілька транзакцій множини $TRRS_i$, аналогічно тому, як було розраховано формулу 2.19 з формули 2.18, отримуємо:

$$p_{l.i0} = \sum_{j \in TRRS_i} \left(e^{-(1-\frac{1}{n})\nu_{l.i} t_{wtl.j}}\right) \frac{\nu_{t.j}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \quad (3.7)$$

2. Визначаються змінні для сум ймовірностей (для кожного ресурсу який пронумерований літерою i – своя i-та сума) і присвоюються відповідним i-тому ресурсу можливостям відсутності конфлікту.

3. Порівнюються тривалості подій одного конфлікту, вибирається найменша тривалість і ймовірність події, цієї найменшої тривалості відповідна додається до відповідної змінної суми.

4. Далі порівнюються тривалості подій більшого на один кількості конфліктів для кожного ресурсу, ніж на попередньому кроці конфлікту, вибирається тривалість, найближча до попередньої тривалості і ймовірність цієї події додається в відповідну суму. Після цього суми перемножуються і отримуємо таке значення Ω_l. Відзначимо, що якщо найближчі довжини обробки різних подій співпадають, то відповідна Ω_l буде дорівнює добутку відповідних сум ймовірностей цих подій. Тобто, оскільки різні тривалості очікування:

$$\Omega_{lj_3} = \prod_{\forall i \in RSTR_{j_3}} \left(\sum_{\substack{k=0 \\ j_1 \in TRRS_i}}^{k t_{wtl.j_1} < d_{lj_3}} \frac{\nu_{t.j_1}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \frac{\left(\nu_{l.i} t_{wtl.j_1}\right)^k}{k!} e^{-\nu_{l.j} t_{wtl.j_1}} \right) \quad (3.8)$$

Відзначимо, що при обчисленні суми по k перебираються всі довжини транзакцій в які залучений ресурс і вибираються такі j_1 та k для яких виконується нерівність $kt_{wtl.j_1} < d_l$. Присутність індексів j_1 ,j_2 та j_3 деякою мірою ускладнює читання, однак, щоб не виникало плутанини (а у нас взаємна залежність середніх тривалостей класу ресурсів від класу транзакцій і навпаки), ми дотримуємося правила, протягом цієї наукової

праці нумерувати класи транзакцій індексом *j* (можливо, як і в цьому прикладі – з номерами), класи ресурсів – індексом *i*, конфлікти – індексом *k*, вузли – індексом *n*, а впорядковані *«можливі тривалості»* індексом *l*.

Відзначимо необхідність нормування *«можливих тривалостей»*. Розглядаючи приклад вище, ми вважали, що кожна наступна довжина кратна довжині транзакції. Однак, перша в черзі заявка в середньому очікує половину тривалості транзакції. Таким чином, при формуванні *«можливих тривалостей»* для транзакції j_2-ого класу кожен наступний варіант тривалості це:

$$d_{lj_2} = t_{wtl.\,j_1}\left(k - \frac{1}{2}\right) \quad \begin{matrix} j_1 \in TRRS_i \\ i \in RSTR_{j_2} \end{matrix} \qquad (3.9)$$

номер *l* буде залежати від того, яким за впорядкуванням виявиться «можлива тривалість», що позначена літерою *d*; d_{lj_2} – це всі кратні *(k-1/2)* тривалості класів транзакцій, що використовують всі класи ресурсів транзакції класу j_2.

Знаючи Ω_{lj_2}, $\Omega_{(l-1)j_2}$ та d_{lj_2} можемо знайти середню тривалість черги для паралельного очікування черги у транзакції класу j_2:

$$t_{queue.\,j_2} = \sum_{l=1}^{\infty} \left(\Omega_{l\,j_2} - \Omega_{l-1\,j_2}\right) d_{lj_2} \qquad (3.10)$$

3.2.3. Знаходження значень ймовірностей p_{rl}, p_{inv}, p_{undo} та коефіцієнту незакешування φ при використанні двофазної обробки транзакції

Нагадаємо, що p_{rl} – це ймовірність того, що довелося звертатися до об'єкту «черга» на іншому вузлі, оскільки в заголовку сторінки було присутнє «блокування» із зазначенням адреси «черги», що їй відповідає, і «черга» виявилася не на локальному вузлі. Ситуація з p_{rl} – досить проста. Оскільки повідомлення об'єкту «чергу» будуть відправлятися паралельно, а

не послідовно, для всіх використовуваних *j*-тою транзакцією ресурсів, справедлива формула:

$$p_{rl.j} = 1 - \prod_{i \in RSTS_j} (1 - p_{rl.i}) \quad (3.11)$$

де $p_{rl.i}$ обчислюється за формулою 2.19 для кожного класу ресурсу (ресурси нумеруються *i*), який використовується *j*-тим класом транзакції.

Ситуація с p_{undo}, p_{inv} та φ – аналогічна, тільки для знаходження $p_{undo.i}$ буде використовуватися формула 2.21, для $p_{inv.i}$ – 2.24, а для φ_i – 2.15. Також $\sigma_{w.i}$ та $\sigma_{w.i}$ відображають що p_{undo} використовується тільки в при читанні ресурсу, а p_{inv} – тільки при внесенні змін до нього.

$$p_{undo.j} = 1 - \prod_{i \in RSTS_j} \sigma_{r.i} (1 - p_{undo.i}) \quad (3.12)$$

$$p_{inv.j} = 1 - \prod_{i \in RSTS_j} \sigma_{w.i} (1 - p_{inv.i}) \quad (3.13)$$

$$\varphi_j = 1 - \prod_{i \in RSTS_j} (1 - \varphi_i) \quad (3.14)$$

Зауважимо, що в дійсності, ймовірність $p_{inv.j}$ – нижче, оскільки очікування суми тривалостей перезапиту сторінки та черги до ресурсу в потоках, які виникли в результаті розпаралелювання, з більш короткими тривалостями черг, ніж максимальна тривалість черги, можуть виконуватися паралельно з очікуванням цієї максимальної черги. Однак в самому песимістичному варіанті – всі тривалості черг однакові – буде саме розрахована за формулою 3.13 ймовірність. Оскільки наше завдання – обґрунтувати ефективність запропонованого методу в порівнянні з традиційним, песимістична оцінка допустима, якщо, звичайно, ефективність буде обґрунтована.

3.2.4. Підсумкова формула тривалості транзакції при використанні двофазної обробки транзакції

Серед складових компонент тривалості транзакції, при двофазної обробці транзакцій додається час на обробку виключення (exception handling) [89], тобто реалізацію коректувальних дій. Для того, щоб оцінити тривалість обробки виключення, необхідно знати залежність ресурсів в транзакції: а саме: чи використовуються дані, що вибиралися протягом попередніх маніпуляцій транзакції в наступних її маніпуляціях. Найчастіше залежність у транзакції – це залежність даних таблиці від індексу. Але по кожному класу транзакції для кожного (*i*-того) класу ресурсу, що входить в цей клас транзакції, необхідно проаналізувати скільки ресурсів протягом подальшої транзакції залежить від попереднього звернення до цього *i*-того ресурсу. Крім того, необхідно оцінити і ймовірність обробки виникнення виключення на *i*-тому ресурсі. Позначимо цю ймовірність $p_{xcp.i}$. З огляду на, що на кожен залежний ресурс, можливо, доведеться перечитати одну сторінку, і всі сторінки при обробці виключення ми також будемо запитувати паралельно – час обробки складе $t_{acc.0}+t_{p.queue}$:

$$t_{xcpt.j}=\left(1-\prod_{\forall i\in RSTS_j}\left(1-p_{xcp.i}\right)\right)\left(t_{acc.0}+t_{p.queue.j}\right) \quad (3.15)$$

Необхідно відзначити, що реальна чергу при виключенні включає не всі ресурси транзакції, а тільки ті, при зверненні до яких на першій фазі були використані некоректні дані. Таким чином – черга $t_{p.queue}$ – це явна оцінка зверху. Однак, щоб не ускладнювати обчислення – скористаємося нею.

Імовірність обробки виключення $p_{xcp.i}$ – це добуток двох її складових ймовірностей. Перша – $p_{xcpinv.i}$ – ймовірність того, що з моменту останнього кешування була зміна сторінки, воно торкнулося саме того рядка, який оброблюється маніпуляцією, яка оброблює ресурс *i*-того класу, і при цьому сторінка виявилася незакешованою. І друга $p_{xcprs.i}$ – ймовірність того, що

використані на першій фазі дані вплинули на обробку ресурсу *i*-того класу, який залежав від ресурсів, що оброблювались попередніми маніпуляціями. Це означає, що досліджуючи залежності ресурсів, ми повинні скласти список ресурсів для яких $p_{xcprs.i}$ ненульова. Відзначимо, що обчислення ймовірності p_{xcprs} можливо тільки після аналізу залежностей у класах транзакцій трафіку.

Тепер оцінимо ймовірність p_{xcpinv}. При виведенні формули 2.15 у нас так само, як і у випадку виведення формули 2.15, один потік кешував заявки, а другий інвалідував. Однак інтенсивність ν_{inv}, що інвалідує – змінилася, а інтенсивність кешування – залишилася такою ж:

$$\nu_{inv}=\left(1-\frac{1}{n}\right)\nu_l \qquad \nu_{cch}=\frac{\nu_r+\nu_w}{n}$$

Тоді, проробивши ті ж дії, що і при виведенні формули 2.15 можемо записати:

$$p_{xcpinv}=\varphi_i\frac{\nu_{l.l}\left(1-\frac{1}{n}\right)}{\left(\nu_{l.i}\left(1-\frac{1}{n}\right)+\frac{\left(\nu_{r.i}+\nu_{w.i}\right)}{n}\right)}=\varphi_i\frac{\nu_{l.i}\left(n-1\right)}{\nu_{l.i}\left(n-1\right)+\nu_{r.i}+\nu_{w.i}} \quad (3.16)$$

або підставивши:

$$p_{xcpinv.i}=\frac{\nu_{w.i}\left(n-1\right)\nu_{l.i}\left(n-1\right)}{\left(\nu_{w.i}\left(n-1\right)+\nu_{r.i}+\nu_{w.i}\right)\left(\nu_{l.i}\left(n-1\right)+\nu_{r.i}+\nu_{w.i}\right)} \quad (3.17)$$

Проаналізувавши формули 3.16 і 3.17 варто відзначити, що в багатьох випадках значення p_{xcpinv} – близькі до нуля, оскільки кількість вузлів рідко перевищує 10, а співвідношення звернень до ресурсу і сторінки, як правило, від 1 до 100. Однак при аналізі кожного трафіку необхідно обчислювати цю ймовірність у випадках, коли в класі транзакцій містяться залежності між ресурсами, та інтенсивності доступу до сторінки і ресурсу суттєво не відрізняються.

Розпишемо послідовно всі затримки транзакції при двофазній обробці:

$$t_{wtl.j}=\varphi_j t_{acc.0}+t_{p.queue.j}+p_{rl.j}t_{rl}+p_{undo.j}t_{acc.0}+ \\ +\left(t_{queue.j}+p_{inv.j}\left(t_{acc.0}+t_{p.queue.j}\right)\right)+p_{xcp.j}\left(t_{acc.0}+t_{p.queue.j}\right) \quad (3.18)$$

Фактично формула дуже нагадує 2.28. Відмінності лише в тому, що замість підсумовування всіх затримок по всіх ресурсах транзакції, ми рахуємо всі затримки один раз і використовуємо консолідовані в рамках *j*-тої транзакції ймовірності $p_{rl.j}$, $p_{inv.j}$ та $p_{undo.j}$ замість ймовірностей $p_{rl.i}$, $p_{inv.i}$ та $p_{undo.i}$ при обробці кожного ресурсу з індексом *i*. Відзначимо, що виконання *undo* для читань, якщо виникла черга, також виконується паралельно з очікуванням черги і перезапитом сторінки. З урахуванням цього перепишемо формулу 3.15:

$$t_{wtl.j}=\left(\varphi_j+p_{inv.j}+p_{xcp.j}\right)t_{acc.0}+\left(1+p_{inv.j}+p_{xcp.j}\right)t_{p.queue.j}+p_{rl.j}t_{rl}+p_{undo.j}t_{acc.0}+t_{queue.j}$$

(3.19)

також як і для для отримання підсумкової системи рівнянь 2.29 зберемо взаємопов'язані формули 3.10, 3.11, 3.12, 3.13 та 3.19 в одну систему рівнянь:

$$\forall j\in TR \rightarrow \\ t_{wtl.j}=\left(\varphi_j+p_{inv.j}+p_{xcp.j}\right)t_{acc.0}+\left(1+p_{inv.j}+p_{xcp.j}\right)t_{p.queue.j}+ \\ +p_{rl.j}t_{rl}+p_{undo.j}t_{acc.0}+t_{queue.j} \\ t_{queue.j}=\sum_{l=1}^{\infty}\left(\Omega_{lj}\left(RSTR_j\right)-\Omega_{l-1\,j}\left(RSTR_j\right)\right)d_{lj}\left(TRTR_j\right) \\ p_{inv.j}=1-\prod_{i\in RSTS_j}\left(\sigma_{w.i}\sum_{\forall j_2\in TRRS_i}\left(e^{-\nu_{l.i}t_{wtl.j_2}}+\nu_{l.i}t_{wtl.j_2}e^{-\left(\nu_{w.i}+\nu_{r.i}\right)t_{queue.j_2}+\nu_{l.i}t_{wtl.j_2}}\right)\frac{\nu_{t.j_2}}{\sum_{j_3\in TRRS_i}\nu_{t.j_3}}\right) \\ p_{rl.j}=1-\prod_{\forall i\in RSTS_j}\left(\sum_{j_2\in TRRS_i}\left(e^{-\left(1-\frac{1}{n}\right)\nu_{l.i}t_{wtl.j_2}}\right)\frac{\nu_{t.j_2}}{\sum_{j_3\in TRRS_i}\nu_{t.j_3}}\right) \\ p_{undo.j}=1-\prod_{i\in RSTS_j}\left(\sigma_{r.i}\sum_{\forall j_2\in TRRS_i}\left(e^{-\left(1-\frac{1}{n}\right)\left(\frac{\nu_{r.i}}{\nu_{w.i}}\nu_{l.i}\right)t_{wtl.j_2}}\frac{\nu_{t.j_2}}{\sum_{j_3\in TRRS_i}\nu_{t.j_3}}\right)\right) \quad (3.20)$$

Коментуючи систему рівнянь 3.20 потрібно відзначити, що величини Ω_{lj}, $\Omega_{(l-1)j}$ та d_{lj}, алгоритми обчислення яких детально було розглянуто в розділі 3.4.2, є функціями від усього набору значень множин $RSTR_j$ (класи ресурсів використовуються в j-тому класі транзакцій) та $TRTR_j$ (класи транзакцій, пов'язаних ресурсами з j-тим класом транзакцій, включаючи сам j-тий клас). В формулі 3.20 не приводиться докладний опис цих функцій, оскільки вони при розв'язанні системи будуть обчислюватися програмно. Аналогічно тому, як і при розв'язанні системи 2.29, пропонується використовувати ітераційну підстановку взаємозалежних значень для вирішення системи рівнянь. Як і у випадку 2.29, якщо довжина черги, що утворюється в j-тому класі транзакцій перевищує довжину цієї транзакції або довжина черги до сторінок будь-якого класу ресурсів перевищує t_{send} – це означає вхід в стан перевантаження.

3.2.5. Затримки транзакції при використанні конвеєризації фаз

При моделюванні конвеєризації фаз необхідно спочатку оцінити ефект від відсутності будь-якої з версій сторінки на вузлі, що оброблює транзакцію. Відзначимо, що множина класів ресурсів j-того класу транзакції розпадається на підмножини, критерієм поділу на які стає входження ресурсу в ту частину транзакції, яка змогла виконатися до події відсутності сторінки або навпаки, тобто якщо на i_1-ом в транзакції ресурсі виникла подія відсутності сторінки, то всі ресурси до i_1 – це одна підмножина, а всі інші складають іншу підмножину. Зауважимо, що подія відсутності сторінки може виникнути в транзакції не один раз, проте для простоти прикладу будемо вважати, що множина розпалася на 2 підмножини: В залежності від того, чи входить ресурс в різні підмножини, вони можуть перетинатися чи ні. У нашому прикладі двічі будуть повторюватися очікування, пов'язані з обробкою *undo*, запитом «черги» і перезапитом сторінки в разі інвалідації. Також двічі буде паралельне

очікування черг до ресурсів, що входять в кожну підмножину. Множину класів ресурсів, що входять у розподілені подіями «відсутності сторінки» інтервали *j*-тої транзакції, позначимо $SRSTR_{sj}$. Індексом *s* позначимо номер цієї підмножини за впорядкуванням виконання в транзакції. Потрібно уточнити, що на скільки підмножин і з якими ресурсами множина $RSTR_j$ розпадається, залежить від того, на яких за впорядкуванням ресурсах транзакції відбудеться подія «відсутність сторінки». Оскільки подія «відсутність сторінки» може бути на кожному з ресурсів, то в разі, якщо транзакція складається з i_{total} ресурсів, має сенс ввести вектор булевих значень Z ({0 | 1}, {0 | 1}, {0 | 1}, { 0 | 1} ... {0 | 1}) довжиною i_{total} для позначення комбінацій подій «відсутність сторінок» в процесі обробки транзакції. Відсортувавши поразрядно цей вектор, можемо отримати індекс-число z, яке однозначно визначає комбінацію «відсутностей сторінки» в даному класі транзакцій. Позначимо $PARTS_{jz}$ множину підмножин $SRSTR_{sj}$, кожна з яких використовується для формування черги, на які розпадається множина $RSTR_j$, в разі *z*-того вектора булевих значень Z. Тоді, враховуючи вищезазначене, черга до ресурсу буде формуватися за алгоритмами, що були запропоновані в розділі 3.4.2, на основі виділених підмножин, які входять до $PARTS_{jz}$, а не множини $RSTR_j$.

Крім того, кількість разів, рівну потужності множини $PARTS_{jz}$, будуть повторюватися очікування пов'язані з обробкою *undo*, запитом «черги» і перезапитом сторінки в разі інвалідації.

Запишемо затримки транзакції при методі двофазної обробки даних у вигляді формули для випадку *z*-того вектора відсутності сторінок:

$$S_{js} \in PARTS_{jz} \rightarrow$$
$$t_{wtl.j} = \sum_{\forall S_s} \left((\varphi_s + p_{inv.s} + p_{xcp.s}) t_{acc.0} + (1 + p_{inv.s} + p_{xcp.s}) t_{p.queue.s} + p_{rl.s} t_{rl} + p_{undo.s} t_{acc.0} + t_{queue.s} \right)$$

(3.21)

Нагадаємо, що s – це лічильник підмножин $SRSTR_{js}$, які утворюють множину $PARTS_{jz}$. В даному прикладі змінна-множина S використовується

для перебору всіх варіантів $PARTS_j$. Роспишемо, чому в цьому випадку будуть рівні значення $p_{rl.s}$, $p_{inv.s}$, $p_{undo.s}$, $t_{p.queue.s}$ та $t_{p.queue.s}$. Відзначимо, що індексація по s у кожному j-тому класі транзакцій своя, але щоб не ускладнювати формули ми не пишемо $p_{rl.js}$, $p_{inv.js}$, $p_{undo.js}$, а також $t_{p.queue.js}$ и $t_{p.queue.js}$:

$$p_{undo.s}=1-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\sigma_{r.i}\left(1-p_{undo.i}\right) \quad (3.22)$$

$$p_{inv.s}=1-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\sigma_{w.i}\left(1-p_{inv.i}\right) \quad (3.23)$$

$$\varphi_s=1-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\left(1-\varphi_i\right) \quad (3.24)$$

Що стосується обчислення $t_{p.queue.s}$ – то система рівнянь 3.5 лишиться без змін, але формула 3.2 для розрахунку p_k – ймовірності k конфліктів, яка використовується в 3.5, у випадку подій «відсутності сторінки» набуває наступного вигляду:

$$p_k(t)=\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\left(\sum_{k_{itl}=0}^{k}\varphi_i\frac{\left(\nu_{pl.i}t\right)^{k_{itl}}}{k_{itl}!}e^{-\nu_{pl.i}t}\right)-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\left(\sum_{k_{itl}=0}^{k-1}\varphi_i\frac{\left(\nu_{pl.i}t\right)^{k_{itl}}}{k_{itl}!}e^{-\nu_{pl.i}t}\right) \quad (3.25)$$

При обчисленні $t_{queue.s}$ – ситуація аналогічна. Тривалість $t_{queue.s}$ залежить від Ω_{ljs}, $\Omega_{(l-1)js}$ та d_{ljs}, які в свою чергу перераховуються по підмножині $PARTS_{jz}$, а не по всьому набору класу ресурсів j-того класу транзакції. Наведемо відповідні формули:

$$\forall S_s\in PARTS_{jz}\,\exists\begin{cases}D_{js}\{d_{1\,js},d_{2\,js},\ldots,d_{ljs},\ldots\}\\ \forall(l_1>l_2)\rightarrow(d_{l_1 s}>d_{l_2 s})\end{cases}\rightarrow\begin{cases}d_{ls}=t_{wtl.j}\left(k-\frac{1}{2}\right)\\ i\in S_s\\ j\in TRRS_i\end{cases} \quad (3.26)$$

$$\Omega_{ls}=\prod_{\forall i\in S_s}\Bigg(\sum_{\substack{k=0\\ j_1\in TRRS_i}}^{k\,t_{wtl.j_1}<d_{ls}}\frac{\nu_{t.j_1}}{\sum\limits_{j_2\in TRRS_i}\nu_{t.j_2}}\frac{\left(\nu_{l.i}t_{wtl.j_1}\right)^k}{k!}e^{-\nu_{l.j}t_{wtl.j_1}}\Bigg) \quad (3.27)$$

А остаточна формула для обчислення $t_{queue.s}$ буде:

$$t_{queue.s}=\sum_{l=1}^{\infty}\left(\Omega_{ls}-\Omega_{(l-1)s}\right)d_{ls} \quad (3.28)$$

Таким чином, система рівнянь для обчислення тривалостей транзакцій набуває вигляду:

$$\begin{gathered}
S_{js}\in PARTS_{jz}\rightarrow\\
t_{wtl.j}=\sum_{\forall S_s}\Big(\left(\varphi_s+p_{inv.s}+p_{xcp.s}\right)t_{acc.0}+\left(1+p_{inv.s}+p_{xcp.s}\right)t_{p.queue.s}+\\
+p_{rl.s}t_{rl}+p_{undo.s}t_{acc.0}+t_{queue.s}\Big)\\
t_{queue.s}=\sum_{l=1}^{\infty}\left(\Omega_{ls}\left(S_{js}\right)-\Omega_{(l-1)s}\left(S_{js}\right)\right)d_{lj}\left(S_{js}\right)\\
p_{inv.j}=1-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\sigma_{w.i}\left(\sum_{\forall j_2\in TRRS_i}\left(e^{-\nu_{l.i}t_{wtl.j_2}}+\nu_{l.i}t_{wtl.j_2}e^{-\left(\nu_{w.i}+\nu_{r.i}\right)t_{queue.j_2}+\nu_{l.i}t_{wtl.j_2}}\right)\frac{\nu_{t.j_2}}{\sum\limits_{j_3\in TRRS_i}\nu_{t.j_3}}\right)\\
p_{rl.j}=1-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\left(\sum_{j_2\in TRRS_i}\left(e^{-\left(1-\frac{1}{n}\right)\nu_{l.i}t_{wtl.j_2}}\right)\frac{\nu_{t.j_2}}{\sum\limits_{j_3\in TRRS_i}\nu_{t.j_3}}\right)\\
p_{undo.j}=1-\prod_{\substack{\forall i\in S_s\\ S_s\in PARTS_{jz}}}\sigma_{r.i}\left(\sum_{\forall j_2\in TRRS_i}\left(e^{-\left(1-\frac{1}{n}\right)\left(\frac{\nu_{r.i}}{\nu_{w.i}}\nu_{l.i}\right)t_{wtl.j_2}}\frac{\nu_{t.j_2}}{\sum\limits_{j_3\in TRRS_i}\nu_{t.j_3}}\right)\right) \quad (3.29)
\end{gathered}$$

Розглянемо тепер, як змінюються затримки при використанні конвеєризації фаз. Оскільки звернення до «черги» відбувається після отримання сторінки, то виконання другої паралельної обробки ресурсів наступного підмножини $SRSTR_{sj}$ починається зі зсувом на $(t_{acc.0} + t_{p.queue.s})$, третій – на $2(t_{acc.0} + t_{p.queue.s})$ і т.д. (Хоча, в разі високих значень ймовірностей

$p_{rl.s}$ зсув буде всього на t_{send}, за рахунок паралельного виконання запиту до «черги» і перших двох звернень, ми, в даному випадку, будемо використовувати песимістичний сценарій для песимістичної оцінки методу). Також, для спрощення обчислень, оцінимо довжини черг до сторінок як t_{send} – тобто максимальною довжиною черги, після якої наступає перевантаження. Таким чином, на кожному наступному оскільки черги починаються зі зсувом $2t_{net} + 2t_{send}$. Наведемо і точнішу формулу зсуву, однак зауважимо, що вона значно ускладнює розрахунки:

$$\Delta_s = (s-1)\,t_{acc.0} + \sum_{s_{itl}=1}^{s-1} t_{p.queue.s_{itl}} \quad (3.30)$$

Знаючи формулу зсуву ми можемо внести зміни в алгоритм розрахунку «можливих тривалостей»:

$$\forall\, S_s \in PARTS_{jz}\, \exists \begin{cases} D_{js}\{d_{0\,js}, d_{1\,js}, d_{2\,js}, \ldots, d_{ljs}, \ldots\} \rightarrow \\ \forall (l_1 > l_2) \rightarrow (d_{l_1 s} > d_{l_2 s}) \end{cases} \rightarrow$$

$$\begin{cases} d_{ls} = t_{wtl.j}\left(k - \frac{1}{2}\right) + (s-1)\left(2t_{net} + 2t_{send}\right) \\ i \in S_s \\ j \in TRRS_i \end{cases} \quad (3.31)$$

Відзначимо, що додався нульовий діапазон. З урахуванням цієї формули (3.26) набуває нового значення і формула обчислення Ω_{lj}, ми знову розраховуємо довжини черг по всій множині ресурсів j-того класу транзакції $RSTR_j$, а не по его підмножинам $PARTS_j$:

$$\Omega_{lj} = \prod_{\forall i \in RSTR_i} \left(\sum_{\substack{k=0 \\ j_1 \in TRRS_i}}^{k t_{wtl.j_1 + s} + (s(i)-1)(2t_{net} + 2t_{send}) < d_{ls}} \frac{\nu_{t.j_1}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \frac{\left(\nu_{l.i} t_{wtl.j_1}\right)^k}{k!} e^{-\nu_{l.j} t_{wtl.j_1}} \right) \quad (3.32)$$

Звернемо увагу на функцію *s(i)*, яка відображає зворотний мапінг класу ресурсу на номер підмножини *s* в транзакції. *s(i)* повертає за ідентифікатором ресурсу номер послідовності підмножини s, а в разі, якщо

ресурс входить в кілька підмножин – всі ці номери послідовностей при переборі повинні бути розглянуті.

З огляду на, що розраховувати суму, для обчислення очікувань ресурсу в транзакції починаємо з 0, кінцева формула для обчислення $t_{queue.s}$ буде:

$$t_{queue.j} = \sum_{l=0}^{\infty} \left(\Omega_{lj} - \Omega_{(l-1)j}\right) d_{lj} \quad (3.33)$$

Наведемо формулу для обчислення тривалості транзакції при конвеєризації фаз. Відзначимо існування тільки однієї черги. Також відзначимо, що ми вважаємо, що затримки p_{rl} ми будемо рахувати один раз, оскільки в інших випадках вони йдуть на фоні очікування або черги або наступних читань сторінок. Крім того, при обробці останньої «порції» ресурсів, в разі відсутності черги на цій останній ітерації буде мати місце суміщення читання сторінок з очікуванням найдовшої черги. Отже, маємо:

$$\begin{aligned} &S_{js} \in PARTS_{jz} \rightarrow \\ &t_{wtl.j} = \left(\varphi_s t_{acc.0} + t_{p.queue.j} + p_{rl.j} t_{rl} + t_{queue.j}\right) + \\ &\qquad + \Omega' \left(p_{inv.s} + p_{xcp.s}\right)\left(t_{acc.0} + t_{p.queue.(s+1)}\right) \end{aligned} \quad (3.34)$$

Ω', – це ймовірність того, що тривалість черги не перевищить тривалість перезапиту сторінки та обробки виключення, що розраховується за формулою схожу на 3.32:

$$\Omega' = \prod_{\forall i \in RSTR_i} \left(\sum_{\substack{k=0 \\ j_1 \in TRRS_i}}^{k t_{wtl.j_1+s} + (s(i)-1)(2t_{net}+2t_{send}) < 4(t_{net}+t_{send})} \frac{\nu_{t.j_1}}{\sum_{j_2 \in TRRS_i} \nu_{t.j_2}} \frac{\left(\nu_{l.i} t_{wtl.j_1}\right)^k}{k!} e^{-\nu_{l.j} t_{wtl.j_1}} \right) \quad (3.35)$$

Резюмуючи, маємо систему рівнянь, яка розв'язується за аналогією з 2.29, 3.20 та 3.29 ітераційно:

$$t_{wtl.j}=\left(\varphi_s t_{acc.0}+t_{p.queue.j}+p_{rl.j}t_{rl}+t_{queue.j}\right)+$$
$$+\Omega'\left(p_{inv.s}+p_{xcp.s}\right)\left(t_{acc.0}+t_{p.queue.s+1}\right)$$
$$t_{queue.j}=\sum_{l=0}^{\infty}\left(\Omega_{lj}\left(RSTR_j\right)-\Omega_{l-1\,j}\left(RSTR_j\right)\right)d_{lj}\left(TRTR_j\right)$$
$$p_{inv.j}=1-\prod_{\forall i\in RSTS_j}\sigma_{r.i}\left(\sum_{\forall j\in TRRS_i}\left(e^{-\nu_{l.i}t_{wtl.j}}+\nu_{l.i}t_{wtl.j}e^{-\left(\nu_{w.i}+\nu_{r.i}\right)t_{queue.j}+\nu_{l.i}t_{wtl.j}}\right)\frac{\nu_{t.j_2}}{\sum_{j_3\in TRRS_i}\nu_{t.j_3}}\right)$$
$$p_{rl.j}=1-\prod_{\forall i\in RSTS_j}\left(\sum_{j_2\in TRRS_i}\left(e^{-\left(1-\frac{1}{n}\right)\nu_{l.i}t_{wtl.j_2}}\right)\frac{\nu_{t.j_2}}{\sum_{j_3\in TRRS_i}\nu_{t.j_3}}\right)$$
$$p_{undo.j}=1-\prod_{\forall i\in RSTS_j}\sigma_{r.i}\left(\sum_{\forall j_2\in TRRS_i}\left(e^{-\left(1-\frac{1}{n}\right)\left(\frac{\nu_{r.i}}{\nu_{w.i}}\nu_{l.i}\right)t_{wtl.j_2}}\frac{\nu_{t.j_2}}{\sum_{j_3\in TRRS_i}\nu_{t.j_3}}\right)\right) \qquad (3.36)$$

3.3. Зміни в математичній моделі, для оцінки затримок запропонованої процедури доступу, що використовує метод призначення обробника ресурсу

Важливо відзначити, що метод призначення обробника ресурсу вносить зміни не в модель конфліктів, а в модель затримок. Це, як було описано в розділі 3.1.2, відбивається на збільшенні часу обробки, проте знімає проблему перевантаження по ресурсу. Розглянемо описаний в розділі 3.1.2 метод в затримках. Пересилання майстер-вузлу запиту і отримання від нього відповіді – це $2t_{net}$. Відправлення «завдання на зміну сторінки» це ще t_{net}, а відповідь хостера з пересиланням актуальної версії сторінки зі зробленими змінами – це t_{send}. Таким чином, загальний час:

$$t_{acc}=3t_{net}+t_{send} \quad (3.37)$$

Оцінимо коефіцієнт кешування. У разі отримання SCUR запиту, якщо сторінка знаходиться в стані «розвантаження», в стан SCUR сторінка не перекладається. Вузлам, що запрошують SCUR пересилається актуальна версія сторінки, але CR. У такому випадку ймовірність кешування означає,

що заявка прийшла на вузол-хостер, і завжди дорівнює *1/n*. Часом стояння в черзі, так як воно не включає в себе часи пересилань, ми можемо знехтувати. Крім того, ймовірність, що вузол виявиться майстер-вузлом сторінки також *1/n*. Якщо майстер і хостер виявляться на одному вузлі, але ймовірність потрапити на вузол не майстер і не хостер дорівнює *(1-1/n)*, а в разі, якщо на різних – *(1-2/n)*. Очевидно, що рознесення майстра і хостера на різні вузли дає зменшення середнього часу відклику.

Відобразимо залежність часу доступу від того, чи є вузол майстром або хостером відображену в таблиці 3.1, з урахуванням позначень зміни станів XCUR та SCUR, прийнятих в моделі затримок, тобто xx – з XCUR в XCUR, xs – XCUR в SCUR, sx – SCUR в XCUR та ss – SCUR в SCUR; h в даному випадку означає, що звернення було до вузла-хостера; m – що до вузла-майстра.

Таблиця 3.1

Тривалість обробки від факторів, що на неї впливають з урахуванням відповідних ймовірностей

	m	h	$\overline{\text{m}}\,\overline{\text{h}}$	Середня тривалість доступу
xx	$t_{net}+t_{send}$	0	$3t_{net}+t_{send}$	$(3t_{net}+t_{send})-\frac{1}{n}(5t_{net}+t_{send})$
sx	-	-	-	-
xs	$t_{net}+t_{send}$	0	$3t_{net}+t_{send}$	$3t_{net}+t_{send}-\frac{1}{n}(5t_{net}+t_{send})$
ss	-	-	-	-

Таким чином, середня тривалість доступу до сторінки за допомогою процедури, що ґрунтується на методі призначення обробника ресурсу:

$$3t_{net}+t_{send}-\frac{1}{n}(5t_{net}+t_{send}) \quad (3.38)$$

Відзначимо, що формула працює і при $n = 2$, незважаючи на те, що вузла не майстра і вузла не хостера немає.

3.4. Висновки по розділу 3

1. Запропоновано метод двофазного виконання транзакції, що дозволяє виключити блокування спільного ресурсу в хмарних СКБД на етапі визначення необхідного набору даних, в результаті чого зменшується тривалість транзакцій і збільшується стійкість систем до перевантажень, пов'язаних з конфліктом за спільні ресурси.
2. Запропоновано метод призначення обробника ресурсу, який за рахунок введення паралельних пересилань дозволяє виключити перевантаження, пов'язані з конфліктом за спільні сторінки.
3. Запропоновано метод конвеєризації фаз виконання транзакції, який, на відміну від послідовного виконання транзакції, дозволяє поєднати виконання окремих частин транзакції, що дозволяє знизити ймовірність перевантаження, пов'язаного з конфліктом за спільні ресурси.
4. Створені й інтегровані в запропоновану в розділі 2 математичну модель методики оцінки тривалостей обробки транзакції і доступу до сторінки для запропонованих методів.

РОЗДІЛ 4. Експериментальна перевірка запропонованої моделі, оцінка очікуваного ефекту від застосування процедур запропонованого способу

4.1. Моделювання трафіків (ґрунтуючись на запропонованій в розділі 2 математичній моделі)

Задача цього розділу – на прикладах трафіків показати придатність запропонованої в розділі 2 математичної моделі для дослідження хмарних СКБД. Крім того, після отримання підтвердження придатності моделі, оцінити ефект від сформульованих в третьому розділі методів, в контексті дослідження стійкості системи до перевантаження і обґрунтувати їх ефективність [91]. Нагадаємо, що запропонована у другому розділі математична модель складається з трьох компонент: моделі затримок, моделі конфліктів і моделі трафіків. І необхідно відзначити ключову роль в цій трійці моделі трафіків, оскільки без результатів моделі трафіків модель конфліктів позбавляється своїх вхідних параметрів. Основна задача моделі трафіків – формалізація досліджуваного трафіку: виділення значущих з точки зору дослідження розподілених СКБД класів ресурсів і формалізація взаємозалежностей ресурсів і транзакцій. Після цього – визначення інтенсивностей звернення до сторінок і ресурсів кожного класу ресурсу. Маючи інформацію про взаємозалежностях транзакцій і ресурсів і знаючи інтенсивності запитування класів ресурсів і сторінок ресурсів цих класів можна вирішити системи рівнянь 2.29, 3.20, 3.29, 3.36, отримані в розділах 2 та 3. Результатом рішення систем рівнянь буде отримання середньої тривалості транзакцій для кожного з класів, що входять в трафік. Крім того, в разі перевищення протягом ітераційного процесу обчислення (2.29, 3.20, 3.29, 3.36) часу очікування сторінки або ресурсу своїх граничних значень, ми можемо констатувати стан перевантаження, тобто зафіксувати граничну інтенсивність, після якої система перестає функціонувати. Дослідження стійкості системи до перевантажень в контексті дослідження хмарних

СКБД представляє набагато більший інтерес, ніж підвищення продуктивності їх роботи.

Для перевірки придатності пропонованої моделі ми візьмемо систему-прототип, і перевіримо відповідність розрахованих математично значень з реально вимірюваними. З огляду на те, що розглянуті трафіки повинні відображати реально існуючі OLTP трафіки, що використовуються в реальному житті, має сенс орієнтуватися на синтетичні тести для оцінки продуктивності OLTP систем. Одним з найбільш поширених тестів є тест TPC-C [68]. Крім того, в роботі розглядаються простіші трафіки, які названі елементарний і трафік проводок [75]. Опис трафіків буде дано нижче. Необхідно розглянути кілька трафіків, щоб показати різну ефективність запропонованих методів в кожному з випадку. Також потрібно відзначити, що у всіх трафіках довжини рядків невеликі, порівняно з розміром сторінки, що визначає кількість рядків на сторінці. Для зручності розрахунків ми будемо вважати, що кількість рядків на сторінці 100, що дуже близько до реальних значень. Однак варіюючи кількістю рядків на сторінці ми можемо показати різну ефективність роботи запропонованих методів, що також буде зроблено. Крім того, важливо показати, що для запропонованих трафіків можна нехтувати середнім часом обробки виключення (exception handling)[89].

4.2. Опис елементарного трафіку (без зворотного зв'язку)

Нехай ми маємо проіндексовану таблицю продуктів. Клієнт спочатку шукає за критеріями продукту в цій таблиці, перевіряючи його наявність на складі (запит SELECT (S)), потім оновлює кількість продукту на складі (запит UPDATE (U)). Якщо кількість продукту на складі 0, клієнт оновлює статистику невиконаних запитів в спеціальній таблиці. Після цього транзакція завершується (фіксація транзакції – операція COMMIT (C)). Будемо вважати, що в 20% випадків на етапі першої вибірки нічого не

знайдено і транзакція завершується, при цьому тільки половина таких запитів (10%) обробляється без звернення до таблиці – тільки індексами, а інші 10% запитів читають з таблиці. Ще в 15% випадків товару на складі не виявляється. Зауважимо, що індекси на таблицю не оновлюється і через деякий час повністю будуть закешовані на кожному вузлі, а значить, пошук через індекс не вимагатиме мережевих пересилань і відповідний час може вважатися 0. Також відзначимо, що індексу по полю «кількість продукту на складі», що оновлюється транзакціями, немає, що характерно для систем, які реалізують склад, оскільки пошук за кількістю товарів, що залишилися рідко вказується у вимогах бізнес-додатків в порівнянні з пошуком за критеріями продуктів.

Нехай середня кількість рядків на сторінці таблиці продуктів 100. Володіючи такою інформацією ми можемо виділити 3 класи транзакцій:

1. Тільки читання таблиці складу (клас ресурсу №1);
2. Оновлення таблиці складу після першого класу транзакцій(читання);
3. Читання таблиці складу і оновлення таблиці невиконаних запитів (клас ресурсу №2) або вставка в неї.

Незважаючи на те, що в транзакції бере участь більша кількість класів ресурсів, закешовані і незмінні індекси (назвемо їх статичними) не створюють вкладу у тривалості обробки транзакцій, і тому при математичному моделюванні не розглядаються. Однак зазначимо, що за їх відсутності поведінка системи буде іншою і потрібно буде по іншому оцінювати часи обробки транзакцій.

Нехай інтенсивність вхідної черги ν. Максимальна інтенсивність читання першого ресурсу дорівнює 0.9ν, оскільки читання присутнє в кожному запиті, крім тих, що оброблюються статичними індексами.

Для другого класу транзакцій ситуація дещо складніша, ніж для третього та першого. Це пов'язане з тим, що ймовірність незакешування для сторінки буде обчислюватись не за формулою 2.15, а з урахуванням

описаної у розділі 2.3 ситуації, коли з запит ресурсів з одного вузла протягом заявки відбувається двічі. В такому разі можемо розписати максимальну і мінімальну інтенсивність читання. Зауважимо, що оскільки читання присутнє у першому, а не у другому класі транзакцій – максимальна і мінімальна інтенсивність читання визначаються для першої транзакції, і будуть, відповідно *0.1v* та *0.75v*.

У другому класі транзакцій, коли запис виконується після читання з того ж вузлу, інтенсивність записів (і мінімальна і максимальна) першого ресурсу – *0.65v*.

Третій клас транзакцій, відповідно, буде мати всі інтенсивності (мінімальні і максимальні до обох ресурсів) рівними *0.15v*.

Середня кількість ресурсів в транзакції (з урахуванням виключення транзакцій, що використовують статичні індекси) – 1.09. Відобразимо ці дані в табл. 4.1, де $v_{r.min}$ – інтенсивність ; $v_{r.min}$ – і; $v_{r.min}$ – і;

Після оцінки відповідних інтенсивностей класів ресурсів транзакцій, що утворюють трафік, необхідно оцінити мінімальний та максимальний час обробки сторінки на кожному кроці транзакції (обробки наступного ресурсу в послідовності транзакції). Розташуємо в табл. 4.2 по вертикалі номера транзакцій, а по горизонталі – якими в транзакції за впорядкуванням будуть ресурси даного класу в певному класі транзакції; число в першому рядку поруч з min і max відображає номер звернення до ресурсу; індекс max при φ – означає, що треба взяти максимальну інтенсивність для читання, та мінімальну для запису, а min – навпаки – мінімальну для читання, та максимальну для запису; індекс при φ – для якого ресурсу розрахований коефіцієнт; $t_{acc.i}(\varphi_{i.max})$ – означає, що при обчисленні $t_{acc.i}$ для певного ресурсу треба брати інтенсивності, що відповідають $\varphi_{i.max}$, а $t_{acc.i}(\varphi_{i.min})$ – відповідно інтенсивності $\varphi_{i.min}$.

Оскільки порівнювати ми будемо з способами, які потенційно повинні бути краще ніж класичний, ми будемо використовувати найменший час,

тобто, на кожному кроці будемо брати мінімальні часи з табл. 4.2. Маючи всі необхідні дані для моделі конфліктів, ми можемо їх підставити в системи рівнянь 2.29, 3.20 і для різних значень ν отримати відповідні середні часи обробки транзакцій. Конвеєризація фаз для даного трафіку не має сенсу, оскільки максимальна кількість ресурсів – 2, що означає, що для систем рівнянь 3.29 і 3.36 підстановка наведених значень не виконуватиметься. У розділі 4.4. результати будуть показані графічно.

Таблиця 4.1

Інтенсивності транзакцій елементарного трафіку.

		Ресурс 1	Ресурс 2	вага
Транзакція №1	$\nu_{r\ .min}/\nu$	0.1	-	75%
	$\nu_{r.max}/\nu$	0.75	-	
	$\nu_{w.min}/\nu$	-	-	
	$\nu_{w.max}/\nu$	-	-	
	ν_l/ν	-	-	
Транзакція №2	$\nu_{r\ .min}/\nu$	0	-	65%
	$\nu_{r.max}/\nu$	0	-	
	$\nu_{w.min}/\nu$	0.65	-	
	$\nu_{w.max}/\nu$	0.65	-	
	ν_l/ν	0.65/100	-	
Транзакція №3	$\nu_{r\ .min}/\nu$	0.15	-	15%
	$\nu_{r.max}/\nu$	0.15	-	
	$\nu_{w.min}/\nu$	-	0.15	
	$\nu_{w.max}/\nu$	-	0.15	
	ν_l/ν	-	0.0015	
Загалом	$\Sigma\nu_{r\ .min}/\nu$	0.25	-	155%
	$\Sigma\nu_{r.max}/\nu$	0.9	-	
	$\Sigma\nu_{w.min}/\nu$	0.65	0.15	
	$\Sigma\nu_{w.max}/\nu$	0.65	0.15	
	$\Sigma\nu_l/\nu$	0.0065	0.0015	

Таблиця 4.2

Мінімальні та максимальні часи обробки ресурсів для елементарного трафіку.

	1min	1max	2min	2max
t1	$t_{acc.1}(\varphi_{1.min})$	$t_{acc.1}(\varphi_{1.max})$	-	-
t2	$(1\text{-}1/n)\ 2t_{net}$	$t_{acc.1}(\varphi_{1.max})$	-	-
t3	$t_{acc.1}(\varphi_{1.min})$	$t_{acc.1}(\varphi_{1.max})$	$t_{acc.3}$	$t_{acc.3}$

Обґрунтуємо в розглянутому трафіку нехтування часом обробки виключень, пов'язаних з оптимістичним підходом методу двофазної обробки транзакції. Залежності між ресурсами в першому і другому класах транзакцій немає, оскільки тільки один ресурс є задіяним в транзакції цього класу. При обробці третього класу транзакцій залежність між першим і другим ресурсами є: якщо при оптимістичному читанні CR версії ресурсу рядки знайдені, а при отриманні актуальної версії сторінки виявилося, що таких рядків уже немає. Тобто $p_{xcprs.i}$ ненульова, однак, підставивши в формулу 3.17 значення v_l, $v_{r.min}$ і $v_{w.min}$, враховуючи, що кількість вузлів – 4 отримуємо значення ймовірності $p_{xcpinv.1}$ менше, ніж *0.02*, що в загальних затримках транзакції, з урахуванням того, що вага третього класу транзакцій – *0.15*, не перевищує *0.003*. Таким чином, допустимість знехтування часом обробки виключення в загальних витратах часу обґрунтована.

4.3. Опис трафіку проводок (трафік зі зворотним зв'язком)

Нехай ми маємо 3 таблиці: таблиця пошуку (search s), таблиця проводок (posting p) і таблиця невиконаних запитів (absent a). Клієнт спочатку шукає продукт в таблиці за критеріями пошуку і його наявність на складі (S_s, S_p), потім ексклюзивно блокує необхідну йому кількість продукту на складі, і перевіряє ще раз наявність знайденого товару на складі, тому що за час читання результатів попереднього запиту продукт міг закінчитися (XS_s, S_p), потім в разі наявності товару складі – виконує

проводку (I_p) (запит INSERT позначимо I) і завершує транзакцію (C-commit – фіксація транзакції), тобто:

$$S_s\ S_p => XS_s\ S_p => I_p => C$$

У разі відсутності продукту на складі, виконується вставка в таблицю невиконаних запитів інформації про продукт, дата та інші атрибути не відбулася покупки. Після чого транзакція завершується (C):

$$S_sS_p => XS_sS_p => I_a => C$$

Для таблиці пошуку маємо ту ж ситуацію, що і для елементарного трафіку, раніше розглянутого в розділі 4.2, тільки замість операції U використовується XS (ексклюзивний SELECT) і, відповідно, будемо розглядати послідовність $S_s => XS_s => C$. Як з точки зору блокування сторінки, так і з точки зору блокування ресурсу, до завершення транзакції поведінка системи – однакова. Єдина відмінність в тому, що в разі XS – на сторінці змінюються лише службові дані, а в разі U – і службові, і дані користувача. Ситуація для таблиці проводок дещо інша. Таблиця проводок має один індекс – індекс первинного ключа. Оскільки час між $S_s\ S_p$ і $XS_s\ S_p$ включає в себе рішення користувача про покупку товару чи ні, і цей час більш ніж на 3 порядки менше часу обробки транзакції, будемо вважати це двома різними транзакціями. Крім того, ми нехтуємо часом фіксації.

Таким чином, ми маємо 3 класи транзакцій:

- Клас транзакцій №1: читання запитаних товарів, клас транзакцій (короткий запис класу S_sS_p).
- Клас транзакцій №2: у разі, якщо читання знайшло необхідні рядки (тобто запитані продукти на складі присутні): ексклюзивне читання таблиці пошуку, внесення змін в індекс первинного ключа таблиці проводок і внесення змін в саму таблицю проводок, клас транзакцій (короткий запис класу $XS_sS_p => I_p$).
- Клас транзакцій №3: читання знайшли рядки, але не знайшли потрібної кількості товарів на складі і вносять зміни в таблицю

невиконаних запитів (короткий запис класу $XS_s\, S_p => I_a$). Відзначимо, що в коротких позначеннях класу, *S* запитує SCUR стан, а *XS*, і *I* – XCUR.

Звернемо увагу на те, що безпосередньо в таблицю пошуку нічого не пишеться – тобто індекси на неї залишаються незмінними. Це означає, що через деякий час всі необхідні сторінки індексів в стані SCUR будуть закешовані на кожному вузлі, і час доступу до цих сторінок буде 0. Але таблиці проводок необхідний первинний ключ, для прискорення пошуку в транзакціях другого класу. Що стосується таблиці невиконаних запитів, то в розглянутому трафіку до неї немає звернень крім вставки (INSERT). Нехай вона також проіндексована одним індексом первинного ключа. Логічно припустити, оскільки дані таблиці заповнюються, але не використовуються трафіком, що розглядається, то ця таблиця буде використовуватися для аналітичних запитів у іншому додатку, що дозволяє її не індексувати. У разі, якщо таблиця невиконаних ресурсів непроіндексована, при вставці кожен вузол зможе вносити свої зміни в сторінки, для яких він сам є майстром, і на які цей вузол володіє XCUR блокуванням. Через деякий час всі вузли будуть мати такі сторінки, і весь процес вставки буде автоматично розпаралелен усіма вузлами. З огляду на те, що вузол, що вносить зміни завжди є майстром та володіє XCUR блокуванням – час доступу до сторінки буде 0. Це чудовий факт. Однак, якщо таблиця проіндексована – всі її індекси в момент поновлення потрібно також оновлювати. Причому, вибірка сторінки індексу для вставки, де розташувати дані, визначається даними, а не наявністю на локальному вузлі XCUR блокування, і факт того, що вузол є майстром для сторінки, також не впливає на вибір сторінки для вставки. Таким чином, якщо таблиця невиконаних запитів проіндексована одним індексом первинного ключа, то часом на вставку в таблицю ми можемо нехтувати, а

обробка вставки в індекс потребує стандартного алгоритму пересилань, описаного у розділі 2.1.

Перелічимо основні класи ресурсів цього трафіку:

- Клас ресурсів №1 – рядки таблиці пошуку;
- Клас ресурсів №2 – входи індексу первинного ключа таблиці проводок;
- Клас ресурсів № 3 – рядки таблиці проводок;
- Клас ресурсів № 4 – входи індексу первинного ключа таблиці невиконаних запитів;
- Клас ресурсів № 5 – рядки таблиці невиконаних запитів. Індекси на таблицю пошуку не змінюються трафіком, оскільки таблиця пошуку не змінюється, і через деякий час будуть повністю закешовані на всіх вузлах.

Отже, ми отримали чіткий опис трафіку проводок. Відзначимо, що він складається тільки з SCUR та XCUR запитів.

Торкнемося співвідношення інтенсивностей класів транзакцій. Нехай 20% запитів не знаходять рядків, що задовольняють умовам пошуку користувача. Однак, в разі проіндексованості таблиці пошуку за всіма основними пошуковими полями, за відсутності необхідних даних ми не звертаємося до таблиці, а всю інформацію отримуємо з індексу. Ще раз відзначимо, що індекси, оскільки таблиця пошуку не змінюється, а лише блокує рядок ексклюзивним запитом, не змінюються. Тобто в 20% випадків, коли рядок не знайдений за параметрами пошуку – сама таблиця пошуку, як і інші основні ресурси не задіюються, а використовуються статичні індекси, закешовані на вузлах. Час відклику в такому випадку можна вважати 0. Також нехай 15% запитів не знаходять продуктів на складі – такі запити оновлюють таблицю невиконаних запитів або вставляють в неї записи, які відносяться до продуктів, якими, можливо, необхідно доповнити склад. З огляду на, що 20% обробляються

звертаючись тільки до статичних індексів, відобразимо в табл. 4.3 інформацію про інтенсивності звернення до нестатичних ресурсів і відповідних їм сторінок

Таблиця 4.3

Інтенсивності ресурсів різних класів транзакцій трафіку проводок при стандартному розмірі сторінки (100 рядків).

		Ресурс 1	Ресурс 2	Ресурс 3	Ресурс 4	вага
Транзакція №1	$v_{r\ .min}/v$	0.8	-	-	-	80%
	$v_{r.max}/v$	0.8	-	-	-	
	$v_{w.min}/v$	-	-	-	-	
	$v_{w.max}/v$	-	-	-	-	
	v_l/v	-	-	-	-	
Транзакція №2	$v_{r\ .min}/v$	-	0	0	-	65%
	$v_{r.max}/v$	-	0.65	0.65	-	
	$v_{w.min}/v$	0.65	0.65	0.65	-	
	$v_{w.max}/v$	0.65	0.65	0.65	-	
	v_l/v	0.0065	0	0	-	
Транзакція №3	$v_{r\ .min}/v$	-	0.15	0.15	-	15%
	$v_{r.max}/v$	-	0.15	0.15	-	
	$v_{w.min}/v$	0.15	-	-	0.15	
	$v_{w.max}/v$	0.15	-	-	0.15	
	v_l/v	-	-	-	0	
Загалом	$\Sigma v_{r\ .min}/v$	0.8	0.15	0.15	-	160%
	$\Sigma v_{r.max}/v$	0.8	0.8	0.8	-	
	$\Sigma v_{w.min}/v$	0.8	0.65	0.65	0.15	
	$\Sigma v_{w.max}/v$	0.8	0.65	0.65	0.15	
	$\Sigma v_l/v$	0.0065	0	0	0	

Відзначимо, що 160% – це не помилка, а вийшло так тому, що на один запит, за нашою класифікацією, виконується 2 транзакції: тобто на кожну з транзакцій ($XS_s\ S_p => I_p\ i\ XS_s\ S_p => I_a$) ми виконуємо ще й транзакцію читання (S_sS_p). Також зауважимо, що інтенсивність блокування в таблиці

4.3 для 2-го і 3-го ресурсу позначена 0. Це не помилка, просто на довжину черги це блокування не впливає. Пов'язано це з тим, що ексклюзивне блокування на першому кроці повністю виключає в цьому трафіку конфлікт за 2й та 3й ресурс. Тобто вся черга збирається на ресурсі таблиці пошуку, а не на наступних ресурсах, що оброблюються транзакцією. Також відзначимо, що при вставці в 4й ресурс – а це індекс первинного ключа таблиці невиконаних запитів, вставка всіх рядків здійснюється в одну сторінку індексу, оскільки індексних входів на сторінці на порядок більше, ніж рядків на сторінці таблиці – будемо вважати, що інтенсивність звернення до ресурсу менше як мінімум в 10 разів інтенсивності звернення до сторінки, але навіть при цьому, доступ до четвертого ресурсу блокується доступом до першого. Таким чином, черга до ресурсу збирається тільки на ресурсах першого класу і інтенсивність звернення до ресурсу четвертого класу на чергу не впливає.

Аналогічно тому, як було зроблено для елементарного трафіку, розташуємо в табл. 4.4 по вертикалі – t1, t2, t3 – класи відповідних транзакцій, а по горизонталі – якими в транзакції за впорядкуванням будуть ресурси даного класу в певному класі транзакції. Число в першому рядку поруч з min і max відображає номер звернення до ресурсу; індекс max при φ – означає, що треба при обчисленні взяти максимальну інтенсивність для читання, та мінімальну для запису, а min – навпаки – мінімальну для читання, та максимальну для запису; індекс i при φ_i – для якого ресурсу розрахований коефіцієнт; $t_{acc.i}(\varphi_{i.max})$ – означає, що при обчисленні $t_{acc.i}$ для певного ресурсу треба брати інтенсивності, що відповідають $\varphi_{i.max}$, а $t_{acc.i}(\varphi_{i.min})$ – відповідно інтенсивності $\varphi_{i.min}$. Оскільки для 4-го і 1-го класів ресурсів мінімальні і максимальні часи доступу не відрізняються – вони відображаються у табл. 4.4 один раз.

Зазначимо, що в цій праці трафік проводок буде розглянуто з двома різними розмірами сторінки: стандартним та зменшеним. Хотілося б

відзначити, що при меншому розмірі сторінки система раніше досягає інтенсивності перевантаження по ресурсам. При експерименті з прототипної моделлю (Oracle RAC) цього можна досягти збільшенням параметра PCTFREE для таблиці до необхідного рівня не дозволяючи системі використовувати при вставках відсоток місця на сторінці, більший, ніж заданий PCTFREE. Можна, також, безпосередньо експериментувати, реально зменшуючи розмір сторінки – але це більш трудомісткий процес. Але найпростіший варіант – оскільки черга до ресурсу 1 керує доступом до ресурсів 1, 2 та 3 – це мати тільки 10 рядків в таблиці проводок.

Таблиця 4.4

Мінімальні та максимальні часи обробки ресурсів для трафіку проводок.

	1min= 1max	2min	2max	3min	3max	4min= 4max
t1	$t_{acc.1}$	-	-	-	-	-
t2	$t_{acc.1}$	$(1-1/n)\ 2t_{net}$	$t_{acc.2}(\varphi_{2.max})$	$(1-1/n)\ 2t_{net}$	$t_{acc.3}(\varphi_{3.max})$	-
t3	$t_{acc.1}$	$(1-1/n)\ 2t_{net}$	$t_{acc.2}(\varphi_{2.max})$	$(1-1/n)\ 2t_{net}$	$t_{acc.3}(\varphi_{3.max})$	$t_{acc.4}$

Наведемо в табл. 4.5 інтенсивності ресурсів при зменшеному розмірі сторінки. Зазначимо, що формули таблиці 4.4 не залежать від розміру сторінки, але значення, які будуть отримані за допомогою цих формул будуть відрізнятися, оскільки інтенсивності впливають на коефіцієнти незакешування φ для всіх ресурсів, та відповідно на $t_{acc.i}(\varphi_{i.max})$.

Оскільки за правилами бізнес-додатка товар нашими транзакціями тільки зменшується на складі, виключення, пов'язані з оптимістичним підходом ($p_{xcprs.i}$ ненульова) можуть виникнути тільки в третьому класі транзакцій. Підставивши в формулу 3.17 значення v_l, $v_{r.min}$ і $v_{w.min}$, враховуючи, що кількість вузлів – 4 отримуємо значення ймовірності pxcpinv.1 для трафіку зі стандартним розміром сторінки менше, ніж 0.02, а для трафіку зі зменшеним розміром сторінки менше 0.2. Може насторожити таке велике значення $p_{xcpinv.1}$ для трафіку з зменшеним розміром сторінки, однак, з огляду на що вага цього класу транзакцій в

загальному трафіку 0.15/1.6 = 0.093, що фактично знімає питання про вплив фактора обробки винятків на час обробки транзакції. Можна було б також вказати на те, що повторний перезапит сторінки і так присутній з ймовірністю p_{inv}, і цей фактор впливає тоді, коли інтенсивності низькі і ймовірність перезапиту p_{inv}, пов'язана очікуванням черги, також низька, але це ускладнює аналіз, оскільки для знехтування достатньо того, що 0.20*0.093<0.02.

Таблиця 4.5

Інтенсивності ресурсів різних класів транзакцій трафіку проводок при зменшеному розмірі сторінки (10 рядків).

		Ресурс 1	Ресурс 2	Ресурс 3	Ресурс 4	вага
Транзакція №1	$v_{r\ .min}/v$	0.8	-	-	-	80%
	$v_{r.max}/v$	0.8	-	-	-	
	$v_{w.min}/v$	-	-	-	-	
	$v_{w.max}/v$	-	-	-	-	
	v_l/v	-	-	-	-	
Транзакція №2	$v_{r\ .min}/v$	-	0	0	-	65%
	$v_{r.max}/v$	-	0.65	0.65	-	
	$v_{w.min}/v$	0.65	0.65	0.65	-	
	$v_{w.max}/v$	0.65	0.65	0.65	-	
	v_l/v	0.0065	0	0	-	
Транзакція №3	$v_{r\ .min}/v$	-	0.15	0.15	-	15%
	$v_{r.max}/v$	-	0.15	0.15	-	
	$v_{w.min}/v$	0.15	-	-	0.15	
	$v_{w.max}/v$	0.15	-	-	0.15	
	v_l/v	-	-	-	0	
Загалом	$\Sigma v_{r\ .min}/v$	0.4	0.15	0.15	-	160%
	$\Sigma v_{r.max}/v$	0.4	0.8	0.8	-	
	$\Sigma v_{w.min}/v$	0.4	0.65	0.65	0.15	
	$\Sigma v_{w.max}/v$	0.4	0.65	0.65	0.15	
	$\Sigma v_l/v$	0.065	0	0	0	

4.4. Моделювання трафіку TPC-C

Оцінюючи трафіки в реальних системах варто орієнтуватися на синтетичні тести, зокрема на TCP-C [68]. У свою чергу трафік TPC-C розпадається на підтрафіки, а критерієм розбиття для цього є клас транзакцій: транзакції нового замовлення (клас транзакцій New-Order, no), транзакції платежів (клас транзакцій payment, pa), транзакції перевірки статусу (клас транзакцій Order-Status, os), транзакції супроводу (клас транзакцій delivery, de), транзакції перевірки складу (клас транзакцій Stock-Level, sl). Параметри співвідношення підтрафіків, допустимі у тесті, а також ті, які було використано в експерименті цієї наукової праці, наведені в табл. 4.6.

Таблиця 4.6

Співвідношення підтрафіків в TPC-C трафіку

№ класу	Клас транзакції	Клас (скор.)	Min % of mix	За фактом %
1	New-Order	no	n/a	31
2	Payment	pa	43.0	50
3	Order-Status	os	4.0	6
4	Delivery	de	4.0	4
5	Stock-Level	sl	4.0	9

Зауважимо, що «Min% of mix» – мінімальні вимоги специфікації тесту до відсотку класу транзакцій, а «За фактом %» – значення інтенсивностей, що були використані протягом моделювання та експерименту.

Задача математичного моделювання трафіку TPC-C тесту – досить громіздка. Почнемо з того, що в транзакціях присутні кілька послідовних звернень до одного ресурсу. Однак у всіх випадках послідовності вибірки і поновлення або оновлень і вибірки відбуваються один за одним. У багатьох реалізаціях SQL (наприклад Oracle і PostgreSQL) ці послідовності можуть бути реалізовані через UPDATE, INSERT або DELETE з конструкцією RETURNING. В такому разі, у всіх транзакціях ресурси обробляються по

одному разу, що значно спрощує моделювання: не потрібно оцінювати мінімальні і максимальні інтенсивності і середні часи звернення до ресурсів. Ми скористуємось можливостями синтаксису RETURNING, і будемо, де можливе його використовувати, для спрощення моделювання.

Наведемо основні класи ресурсів:

- Ресурс №1: таблиця customer – використовується у всіх транзакціях крім транзакцій класу «stock level». Зауважимо, що в процесі трафіку поля пошуку таблиці не оновлюються. Тобто індекси цієї таблиці, що використовуються для пошуку – статичні, і через деякий час вони будуть закешовані на кожному вузлі. А це означає, що в нашій моделі при зверненні до таблиці customer обробляється один ресурс – сама таблиця customer.
- Ресурс №2: таблиця warehouse – використовується в транзакціях класів «New Order» і «Payment». Також відзначимо, що є необхідність тільки в одному індексі (w_id) і він не оновлюється, тобто – статичний, а значить, час обробки індексів таблиці warehouse в класах транзакцій no та pa – 0.
- Ресурс №3: таблиця district. Оскільки двостовбцовий індекс на поля (d_id, d_w_id) таблиці district не змінюється трафіком, тобто – статичний, ми його також ресурсом не вважаємо.
- Ресурс №4 та ресурс №5: індекс первинного ключа таблиці orders (поле o_id) та сама таблиця orders.
- Ресурс №6: таблиця new_order, організована як index organized table [84] (поля NO_W_ID, NO_D_ID, NO_O_ID).
- Ресурс №7: таблиця stock (індекс на поля (s_w_id,s_i_id) – статичний).
- Ресурс №8 і ресурс №9: індекс на таблицю order_line (поля ol_o_id, ol_d_id і ol_w_id) і таблиця order_line.
- Ресурс №10: таблиця item. Зауважимо, що оскільки таблиця item в

транзакціях тільки читається, ми також можемо вважати цей ресурс статичним і час доступу до нього нульовим.

- Ресурс №11: таблиця history. Оскільки ніде в трафіку не здійснюється пошук по цій таблиці – будемо вважати її непроіндексованої.

Запишемо послідовність використання ресурсів в транзакціях в табл. 4.7: *w* означає маніпуляція була записом, а *r* – читанням.

Таблиця 4.7

Взаємозв'язок ресурсів і транзакцій в TPC-C трафіку

Клас транзакції			Номер класу ресурсу								
			1	2	3	4	5	6	7	8	9
no	1	30%	1r	2r	3w	4w	5w	6w	7w	8w	9w
pa	2	50%	2w	3w	1w	10w	-	-	-	-	-
os	3	6%	1r	4r	5r	8r	9r	-	-	-	-
de	4	4%	6w	4r	5w	8r	9w	1w		-	-
sl	5	10%	3r	7r	8r	9r	-	-	-	-	-

Оскільки всі ресурси запитуються в кожній транзакції по 1 разу мінімальних і максимальних часів і інтенсивностей немає. Таким чином, розрахувавши інтенсивності читання і запису сторінок, а також інтенсивність оновлення ресурсів ми зможемо визначити час виконання транзакції і граничні інтенсивності трафіку. Наведемо в табл. 4.8 для кожного з 9 ресурсів інтенсивності читання і запису сторінок, а також інтенсивність оновлення ресурсу. Відзначимо, що ресурс 11 не включений в табл. 4.7 та 4.8, оскільки час на вставку в непроіндексовану таблицю нульовий, а ресурс 11 зустрічається тільки в транзакціях класу no, які тільки вставляють дані в цю таблицю. Крім того буква *i* у полі «Рядків на сторінці ресурсу» означає, що ресурс – нестатичний, і за ним буде звернення до таблиці, яку він індексує. Це означає, що тривалість очікування черги до індексу включена до тривалості очікування черги до ресурсу.

Таблиця 4.8

Інтенсивності класів ресурсів в кожному класі транзакцій в TPC-C трафіку

№ ресурсу		1	2	3	4	5	6	7	8	9	10
Рядків на сторінці ресурсу		100	2	20	і	100	100	100	і	100	100
1(no) 0.3	v_r/v	0.3	0.3	-	-	-	-	-	-	-	-
	v_w/v	-	-	0.3	0.3	0.3	0.3	0.3	0.3	0.3	-
	v_l/v	-	-	0.015	0	0	0	0.003	0	0	-
2(pa) 0.5	v_r/v	-	-	-	-	-	-	-	-	-	-
	v_w/v	0.5	0.5	0.5	-	-	-	-	-	-	0.5
	v_l/v	0.005	0.25	0.025	-	-	-	-	-	-	0
3(os) 0.06	v_r/v	0.06	-	-	0.06	0.06	-	-	0.06	0.06	-
	v_w/v	-	-	-	-	-	-	-	-	-	-
	v_l/v	-	-	-	-	-	-	-	-	-	-
4(de) 0.04	v_r/v	-	-	-	0.04	-	-	-	0.04	-	-
	v_w/v	0.04	-	-	-	0.04	0.04	-	-	0.04	-
	v_l/v	0.0004	-	-	-	0.0004	0	-	-	0.004	-
5(sl) 0.1	v_r/v	-	-	0.1	-	-	-	0.1	0.1	0.1	-
	v_w/v	-	-	-	-	-	-	-	-	-	-
	v_l/v	-	-	-	-	-	-	-	-	-	-
Загалом	$\Sigma v_r/v$	0.36	0.3	0.1	0.3	0.06	0	0.1	0.2	0.16	-
	$\Sigma v_w/v$	0.54	0.5	0.8	0.1	0.4	0.34	0.3	0.04	0.04	-
	$\Sigma v_l/v$	0.0054	0.25	0.04	-	0.035	0	0.003	0.004	0.004	-

Необхідно відзначити, що в разі некоректно введених користувачем даних на вхід наших транзакцій кожен з підтрафіків (no, pa, os, de, sl) знов можно розділити на підтрафіки: підтрафік без помилкових даних, підтрафік з помилкою після першого запиту, підтрафік з помилкою після другого запиту і т. д. Принципово це не змінює суті моделювання, проте ускладнює його. Так підтрафік «no» розпадається на 7 підтрафіків, «pa» на

3 і т. д.. Ми будемо вважати для спрощення моделювання, що всі дані користувачів вводяться вірно, хоча відзначимо, що і в реальному додатку можна реалізувати таку систему перевірок, яка зведе відсоток транзакцій з невірно введеними даними до величин на 2 порядки менше коректно оброблюваних транзакцій.

Таким чином, ми знову маємо повну інформацію для підставлення в системи рівнянь 2.29, 3.20, 3.29 і 3.36.

4.5. Експериментальне порівняння запропонованої моделі з прототипом

Для перевірки адекватності запропонованої моделі реальному об'єкту була розгорнута система Oracle RAC, що складається з 4 вузлів. Всі вузли фізично були розгорнуті на одному фізичному сервері на платформі VMware, яка дозволяє емулювати спільний диск. Для забезпечення співвідношення часу обробки до часу пересилання (повинно бути набагато менше) будемо використовувати утиліту wondershaper, знизивши швидкість передачі між вузлами до 600 KB/s командою "wondershaper vmnet1 600 600". П'ятої віртуальною машиною, також розгорнутої на платформі VMware, буде генеруватися трафік з фіксованою інтенсивністю в 32 потоки. В процесі експерименту ми змінюючи інтенсивність генератора вимірюємо загальний час виконання 1000000 транзакцій, порціями по 10000. Порції по 10000 згруповані в PL/SQL збережених процедурах, для зменшення впливу часу пересилання між клієнтом та сервером. У TPC-C трафіку співвідношення інтенсивностей транзакцій до сумарної інтенсивності зафіксовані так як: New-Order – 30%, Payment – 50%, Order-Status – 6%, Delivery – 4%, Stock-Level – 10%.

На графіках (Рис. 4. та 5.) показані значення, що отримані в результаті моделювання та зібрані в протягом експерименту: тобто розрахований і виміряний час обробки на Oracle RAC. Для зручності відображення

наводяться нормовані величини: по осі абсцис використовується інтенсивність ν трафіку, якою ми варіюємо, помножена на час одного пересилання t_{send}, а по осі ординат – час розділений на час одного пересилання. Таким чином, по осі ординат відображено час в пересиланнях (кількість пересилань на графіку позначена *N*), а по осі абсцис – безрозмірна величина $t_{send}\nu$ – тобто кількість заявок виміряна в t_{send}^{-1} (позначена *ν*).

Графіки залежності тривалості обробки транзакції від інтенсивності (Рис. 4 та 5) побудовані на основі формули 2.29. В експерименті використовувався розглянуті в розділах 4.2, 4.3, 4.4 елементарний трафік, трафік проводок зі стандартним розміром сторінки (100 рядків), графік проводок з зменшеним розміром сторінки (10 рядків) та синтетичний тест TPC-C, який використовується для оцінки продуктивності СКБД.

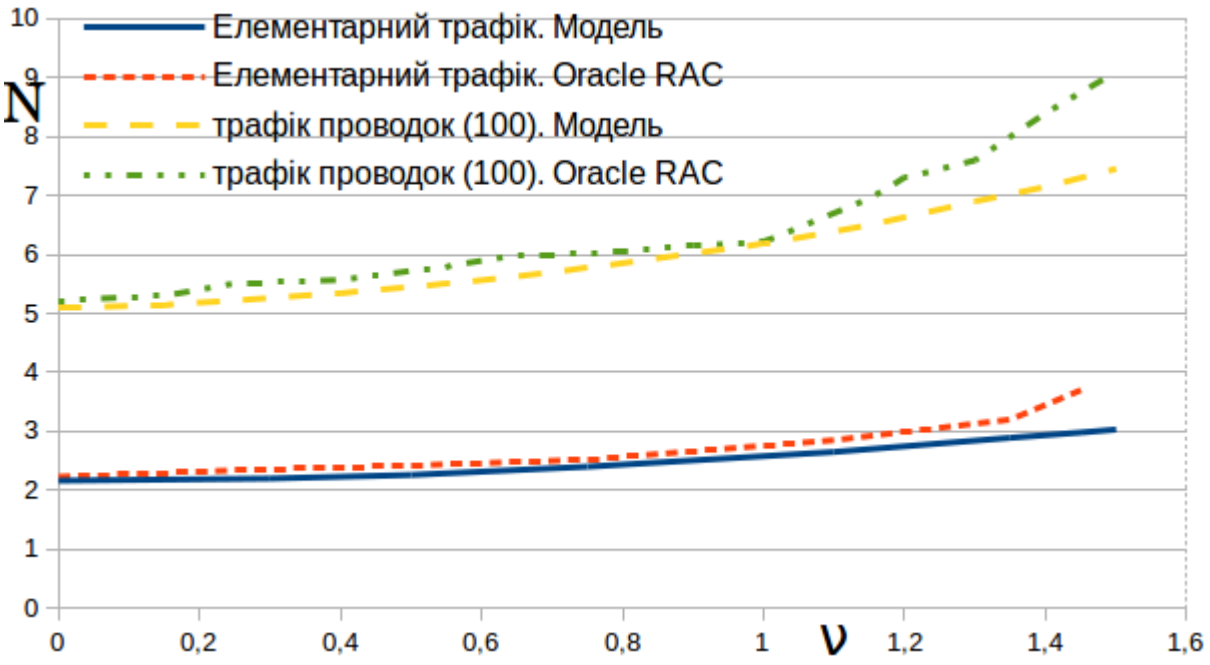

Рис. 4. Змодельовані та фактичні затримки обробки транзакції для елементарного трафіка та трафіка проводок зі стандартним розміром сторінки

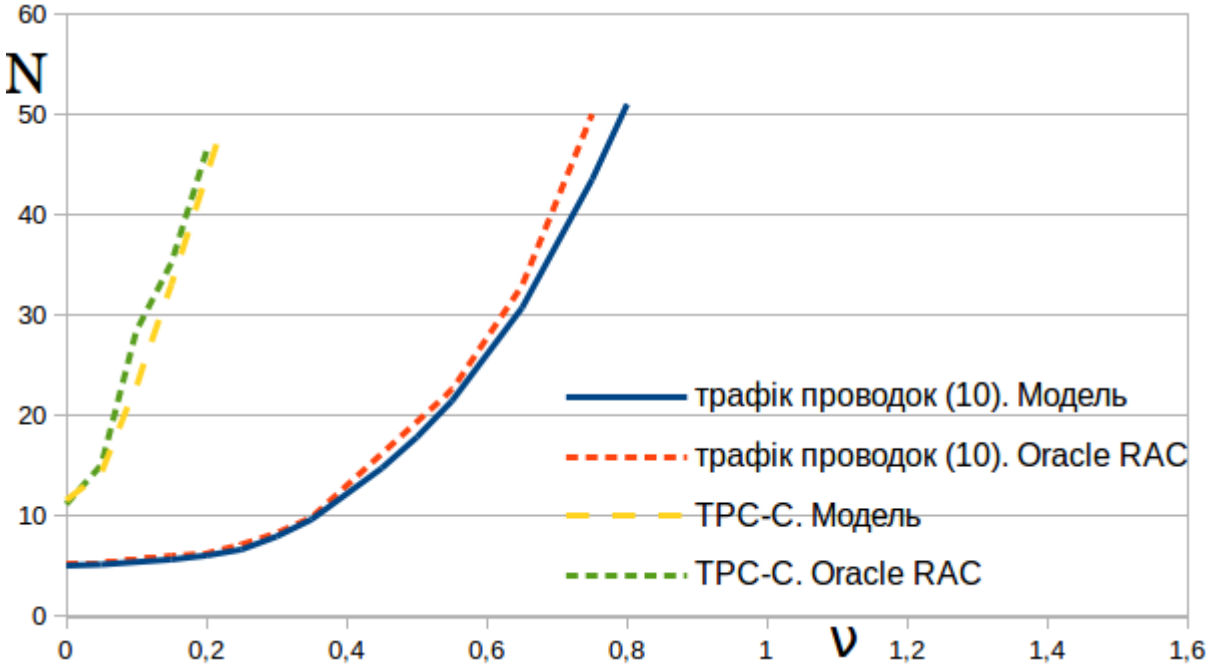

Рис. 5. Змодельовані та фактичні затримки обробки транзакції для TPC-C трафіка и трафіка проводок зі зменшеним розміром сторінки

Необхідно відзначити, що вноситься похибка носить не випадковий, а систематичний характер, оскільки при моделюванні нехтується часом обробки даних в порівнянні з часом пересилання, а також затримками при конфліктах за внутрішні ресурси. Під внутрішніми ресурсами в даній роботі розуміється чергу до центрального процесора, а також черги до внутрішніх конструкцій Oracle, що забезпечує цілісність даних та коректність роботи системи. З огляду на систематичний, а не випадковий характер похибки, що вноситься факторами, які не розглянуті моделлю, а також того факту, що системи рівнянь, які використовуються для визначення тривалостей черг та транзакцій, отримані аналітичним шляхом, а не на основі обробки експериментальної інформації, застосовування критеріїв Фішера або Стьюдента не має сенсу. Для оцінки адекватності достатньо оцінки середньої відносної похибки. Як видно з графіка, перехід до стану перевантаження (тобто різке зростання часу обробки при збільшенні інтенсивності запитів) в реально-функціонуючій

системі відбувається завжди раніше передбаченого значення. Це обумовлюється в першу чергу збільшенням часу обробки конфліктів за спільні ресурси (яке моделлю ігнорується) при збільшенні інтенсивності.

У разі трафіку проводок та TPC-C трафіку реальна інтенсивність переходу в перевантаження відрізняється від передбаченої моделлю не більше ніж на 10%, а в разі елементарного трафіку перевантаження виникає не від того, що час обробки транзакції перевищила поріг переходу в колапс по ресурсам, а від того, що конфлікт за гарячі сторінки (а в елементарному трафіку на транзакцію доводиться обробка 1 сторінки, а з урахуванням даних UNDO – двох сторінок) виникає раніше. В першу чергу, така ситуація є наслідком ескалації конфлікту за ресурси на рівень сторінок. Реально заміряна інтенсивність, після якої система переходила в перевантаження, перевищувала інтенсивність, отриману за допомогою системи рівнянь 2.29, за умов переходу в перевантаження, що були сформульовані в розділі 2.2.4, не більше ніж на 10%. У табл. 4.9 наведені середні значення відносної похибки в залежності від обраного діапазону значень нормованої (кількість заявок виміряна в t_{send}^{-1}) інтенсивності.

Таблиця *4.9*

Середня відносна похибка в залежності від трафіку і діапазону

Трафік	Діапазон	Середня відносна похибка
елементарний	0-0.11	6.8%
	0.5-1	5.4%
	1-1.5	9.9%
Проводок (100)	0-0.5	2.9%
	0.5-1	3.5%
	1-1.5	9,8%
Проводок (10)	0-0.4	4.4%
	0.4-0.8	6.1%
TPC-C	0-0.25	6.3%

Зауважимо, що умовами входу в перевантаження в реальній системі було різке (більш ніж в 10 разів) збільшення середнього часу обробки транзакції, порівняно з часом, отриманим при дослідженні попереднього значення інтенсивності.

4.6. Оцінка ефекту запропонованих методів за допомогою створеної моделі.

Підставивши значення таблиць отриманих в розділах 4.2, 4.3 та 4.5 в формули 2.29 і 3.20 ми можемо порівняти ефективність методу двофазної обробки транзакції в порівнянні з традиційною обробкою транзакції в *shared everything* архітектурі для розглянутих чотирьох трафіків. Також, як і для порівняння з прототипної моделлю, по осі ординат – час в пересиланнях, а по осі абсцис – приведена до t_{send} інтенсивність. Оцінюючи результати порівняння на графіках (рис. 6 та 7) потрібно відзначити, що для двох розглянутих трафіків нарощування інтенсивності зіткнулося з обмеженнями перевантаження по сторінках, а для двох інших – з перевантаженням по ресурсам.

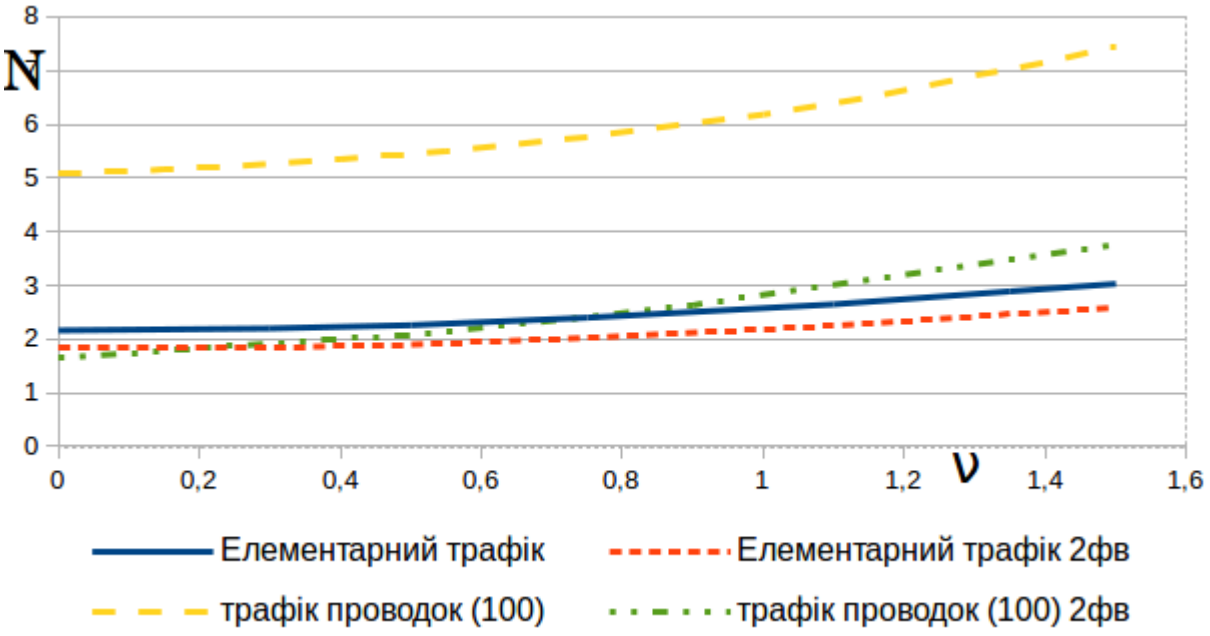

Рис. 6. Метод двофазного виконання транзакції. Відсутність ефекту

У разі перевантаження пов'язаного з конфліктом за сторінки (рис. 4.3), ми, незважаючи на істотне (в разі трафіку проводок зі стандартним розміром сторінки) зменшення часу транзакції не змогли подолати наведену інтенсивність рівну 1.5. Проте для інших двох трафіків можемо спостерігати збільшення можливих інтенсивностей приблизно на 35% та в 3 рази. Відзначимо, що при моделюванні в формули 2.29 і 3.20 будо підставлено час t_{send} рівний t_{net}. Таке припущення забезпечило моделювання глобальної мережі, де час пересилання пакета не залежить від розміру цього пакета. Кількість вузлів, так само, як і при зіставленні з прототипом, приймемо 4.

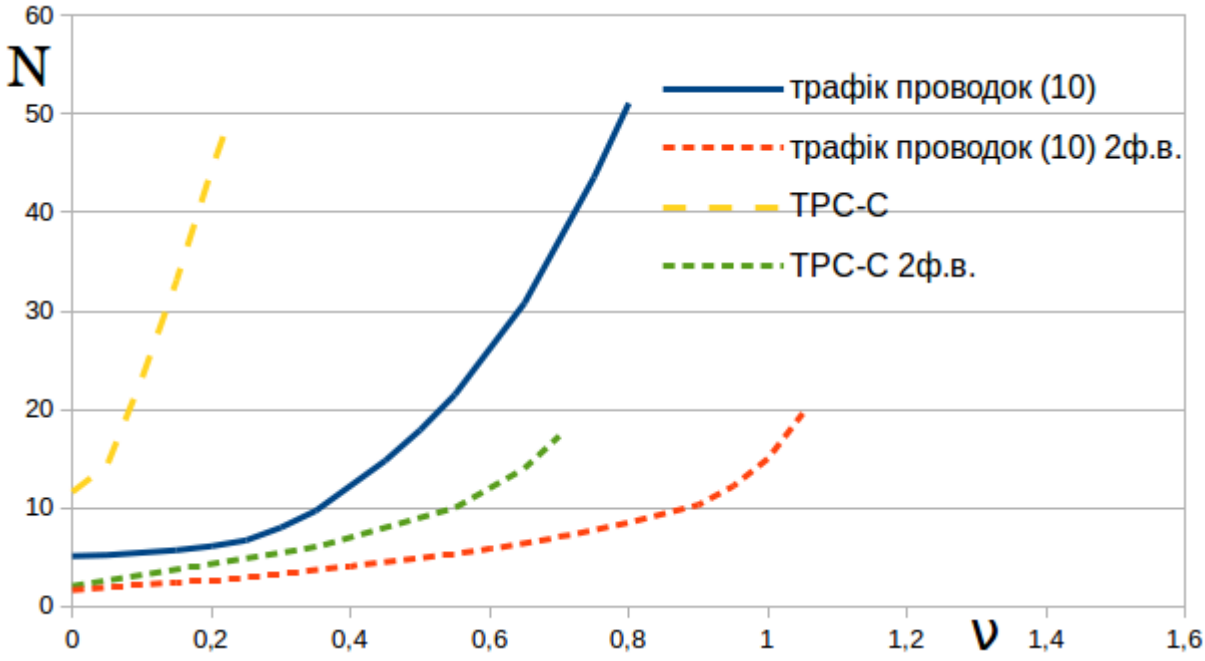

Рис. 7. Метод двофазного виконання транзакції. Більш ніж 3-кратний ефект

Оцінюючи відсутність ефекту на першому графіку важливо згадати про метод призначення обробника ресурсу, запропонований в розділі 3.3.2. Результати моделювання методу призначення обробника ресурсу наведені на рис. 8 та 9.

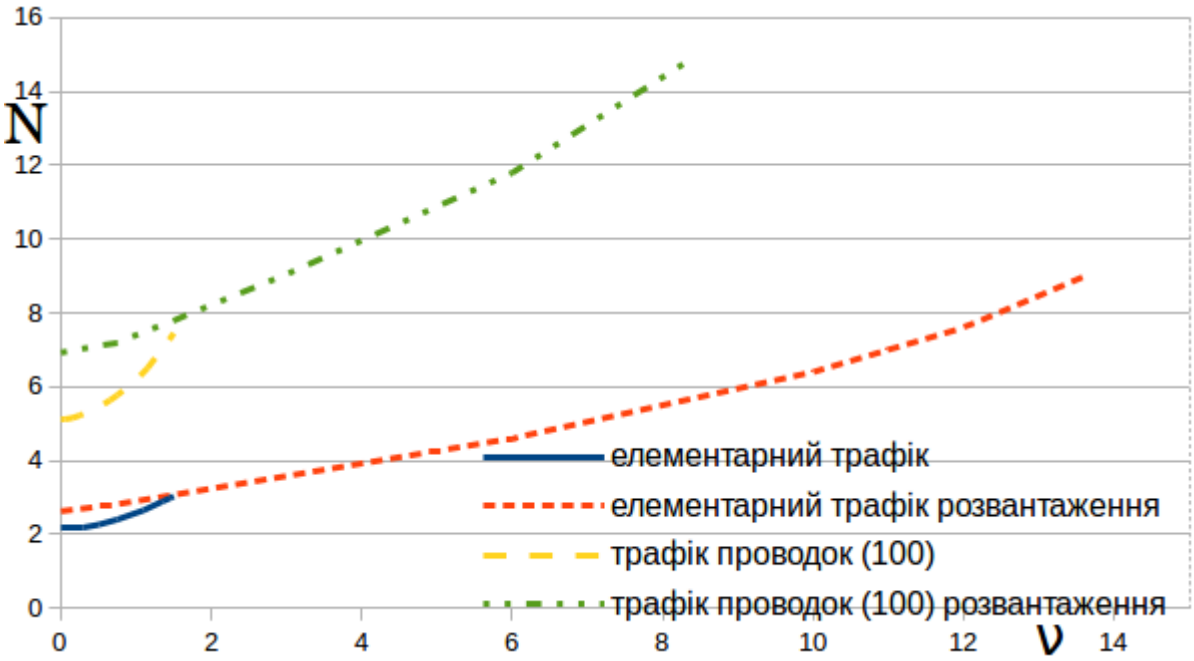

Рис. 8. Метод призначення обробника ресурсу: ефект розвантаження черги

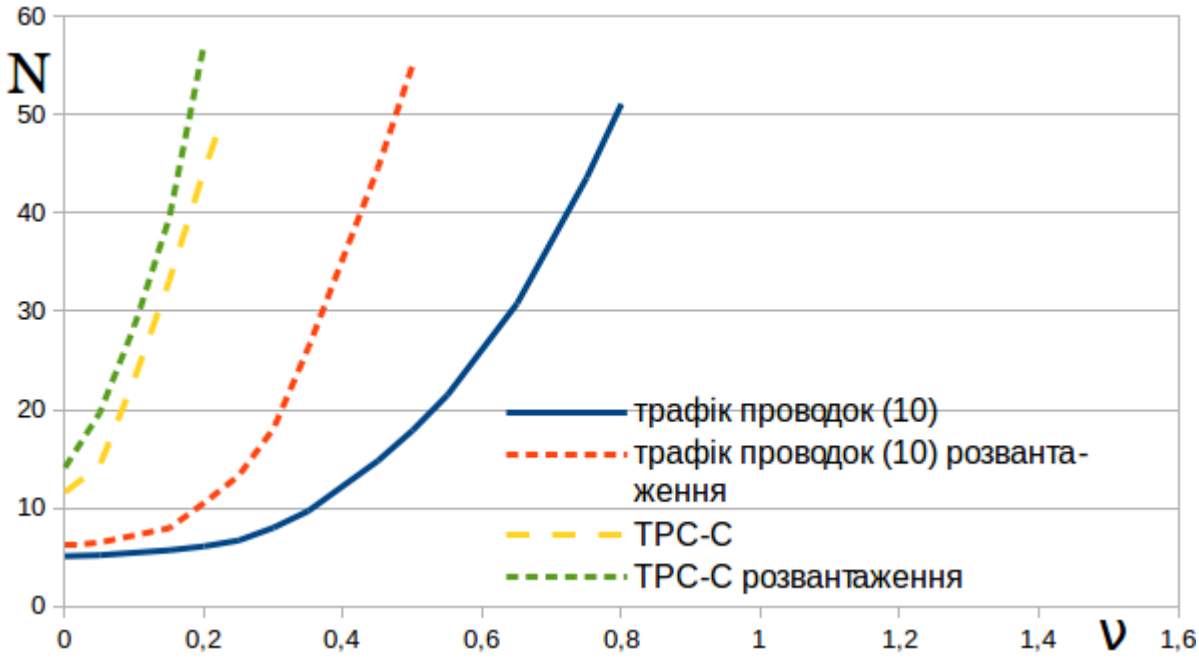

Рис. 9. Метод призначення обробника ресурсу: ефект погіршення

На другому графіку (рис. 9) ми спостерігаємо тільки збільшення тривалості транзакції, і як наслідок – меншу граничну, до переходу в перевантаження, інтенсивність трафіку. На першому графіку (рис. 8) видно, що хоч тривалості часів обробки і стали більше, однак дозволяють

подолати перевантаження по сторінках, що виникає при наближенні до інтенсивності *1.5*, і виграш вийшов приблизно в *10* разів, тобто на порядок. Таким чином, констатуємо, що метод двофазної обробки транзакції ефективний для подолання перевантаження по ресурсам, а метод призначення обробника ресурсу – для подолання перевантаження по сторінках.

Також на рис. 10 наводимо результати моделювання конвеєризації фаз. Відзначимо, що з огляду на існування всього однієї черги у всіх розглянутих трафіках крім TPC-C, сенс застосовувати конвеєризацію фаз має тільки в трафіку TPC-C. Хоча необхідно зазначити, що левова частка затримок в TPC-C трафіку припадає на оновлення другого ресурсу у другій транзакції – оновлення часу останнього доступу до одного з двох складів.

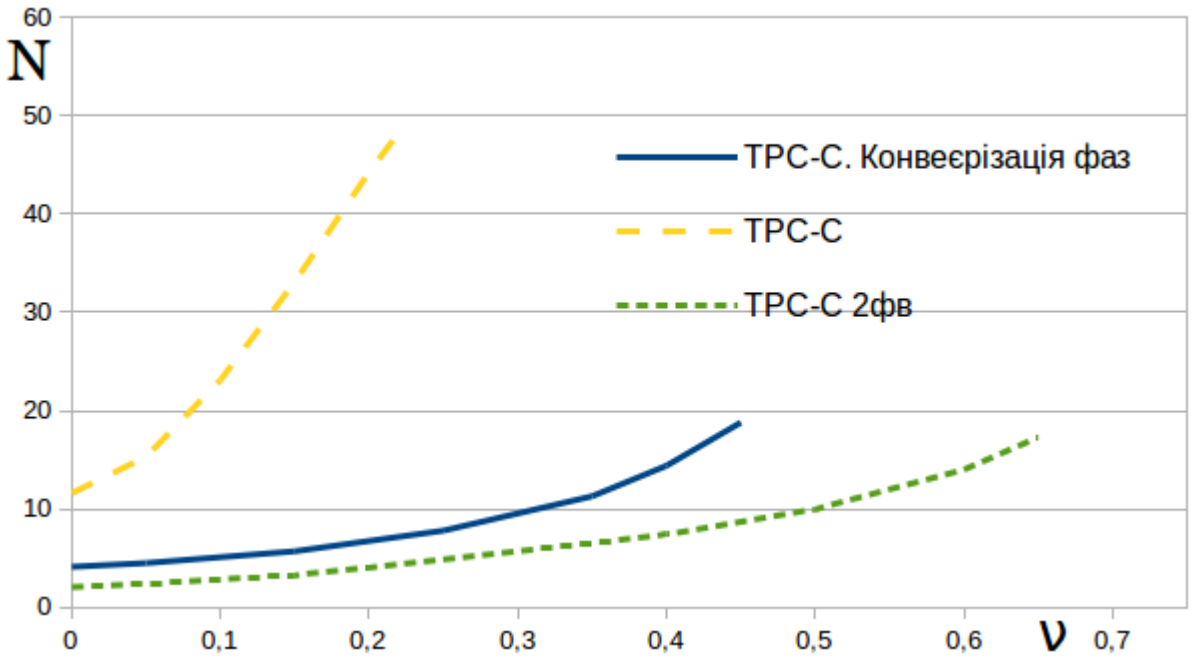

Рис. 10 Конвеєризація фаз. Середня тривалість транзакції при одному випадку відсутності сторінки в кеші

Конвеєризація фаз є природним розвитком методу двофазної обробки транзакції, і графік відображає ефект від цієї новації. Зауважимо, що при більш високих можливостях кешування ефект від конвеєризації буде істотно нижче.

Крім того, очевидною є ідея доповнювати запропоновані методи один одним. Реалізацією цієї ідеї є комбінований підхід включає в себе основні переваги всіх методів в залежності від інтенсивності. При низьких інтенсивностях трафіку класичний підхід доцільніше, а при наближенні до інтенсивностям переходу в перевантаження по ресурсам або сторінок буде обраний або метод двофазної обробки транзакції або призначення обробника ресурсу. Відзначимо, що серед розглянутих трафіків тільки трафік проводок зі стандартним розміром сторінки може бути поліпшений обома методами: спочатку розвантаженням черги за допомогою призначення обробника, а потім двофазним виконанням. На рис. 11 наведено графік комбінованого підходу для цього випадку.

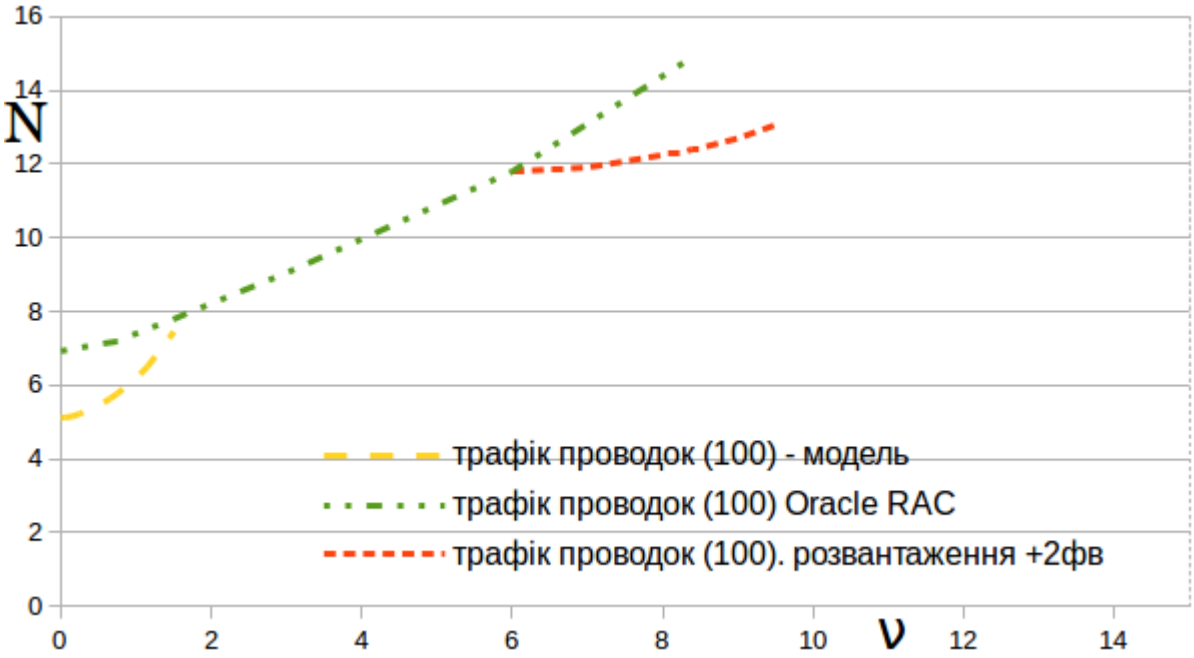

Рис. 11. Комбінований підхід до вибору процедури доступу

З графіка видно, що застосування методу двофазного виконання транзакції після призначення обробника спільного ресурсу дає можливість наростити інтенсивність ще виграш близько 15% після застосування методу призначення спільного ресурсу. Таким чином сумарні можливості системи збільшаться для наведеного прикладу в 12 разів, що може

обґрунтувати використання високошвидкісних глобальних мереж в якості транспорту в хмарної системі.

Зауважимо, що у всіх випадках оцінка можливого ефекту, а вона в модель закладена песимістична, була від 35% до 913%. Такі цифри створюють передумови для використання в якості транспорту глобальних мереж, але слід зазначити, що характеристики трафіку можуть в значній мірі вплинути на ймовірність перевантаження. У той же час, чисто інженерне рішення – підвищити розмір сторінки для трафіку проводок, підвищує ефективність способу з 35% до 650% (комбінований підхід). Наявність гарячого ресурсу в TPC-C трафіку (а це таблиця warehouse, тобто склад, яка містить тільки 2 рядки і оновлюється в 50% транзакцій, в яку вноситься час останнього оновлення складу) робить проблематичним масштабування в хмарному середовищі. Але навіть для такого випадку пропонований спосіб продемонстрував більш ніж 300% виграш в порівнянні з традиційним. Таким чином, запропонований спосіб може зайняти чинне місце в арсеналі засобів масштабування хмарних систем.

4.7. Висновки по розділу 4.

1. На підставі порівняння моделі з системою прототипом в результаті оцінки відносної похибки зроблено висновок про адекватність моделі.
2. Обґрунтовано ефективність методу двофазної обробки транзакції.
3. Обґрунтовано ефективність методу призначення обробника ресурсу.
4. Обґрунтовано ефективність методу конвеєризації фаз.
5. Запропоновано комбінований підхід, в основі якого лежить вибір найкращої, в залежності від характеристик трафіку, комбінації запропонованих і традиційних процедур доступу і вирішення конфліктів, що дозволяє взаємно підсилити ефективність методів запропонованого способу в контексті мінімізації проблеми перевантаження.

Висновки

У дисертаційній роботі виконано теоретичне обґрунтування і отримано нові рішення задачі організації доступу до спільного ресурсу. Для цього досліджені найбільш поширені системи, і, в якості прототипу моделі розподіленої хмарної СКБД, було обрано Oracle Real Application Clusters. Проаналізувавши недоліки прототипної моделі, були запропоновані нові методи організації спільного доступу, які знижують ймовірність перевантажень, пов’язаних з обробкою спільних ресурсів. Основні наукові і практичні результати полягають у наступному.

1. Виконано аналіз існуючих способів доступу до спільних ресурсів в середовищі хмарних обчислень, виявлені недоліки цих способів в контексті вирішуваних задач.

2. Запропоновано і обґрунтовано математичну модель організації одночасного доступу до спільного ресурсу в системах хмарних обчислень, що дозволяє оцінити ефективність відомих методів доступу і виявити причини, що зумовлюють виникнення перевантажень.

3. Вперше запропоновано і обґрунтовано метод двофазної обробки транзакції, який на відміну від відомих методів доступу, дозволяє виключити блокування спільного ресурсу в хмарних системах на етапі визначення необхідного набору даних. Це дозволяє істотно збільшити пропускну спроможність хмарних СКБД і скоротити час відклику.

4. Вперше запропоновано і обґрунтовано метод призначення обробника ресурсу, який за рахунок введення паралельних пересилань дозволяє виключити перевантаження, пов’язане з конфліктом за спільні сторінки.

5. Запропоновано спосіб конвеєризації фаз виконання транзакції, який дозволяє знизити ймовірність перевантаження за рахунок поєднання виконання окремих частин транзакції.

6. Запропоновано і обґрунтовано комбінований підхід до вибору процедури доступу до спільних ресурсів, який в залежності від інтенсивності транзакцій і кількості оброблюваних ресурсів в транзакції дозволяє оптимізувати процедуру спільного доступу.

7. Проведено експериментальну перевірку запропонованої моделі та зроблена оцінка очікуваного ефекту від застосування розроблених методів.

Список використаних джерел

1. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций /Гусев Е.И. // - Вісник НТУУ "КПІ". Сер. Інформатика, управління та обчислювальна техніка. - 2012. Випуск 57. - С.76 – 80

2. MapReduce: внутри, снаружи или сбоку от параллельных СУБД? /Кузнецов, Сергей// - Труды Института системного программирования РАН. – : Институт системного программирования РАН, 2010. – Т.19. – С.35 – 40. – ISSN 2079-8156.

3. SQL or NoSQL that is the Question /Berndt D.J., Lasa R., McCart J. // - SiteWit Corporation: University of South Florida, 2012, p. 14

4. НЕДОСТАТКИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ / А.В. Бурмистров Ю.С. Белов // ЭЛЕКТРОННЫЙ ЖУРНАЛ: НАУКА, ТЕХНИКА И ОБРАЗОВАНИЕ , 2015 3 (3)

5. От SQL к NoSQL и обратно. / Селезнев К. // Открытые системы. СУБД. 2012. No 2. С. 28.

6.RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's./ Padhy R.P., Patra R.M., Satapathy S.C.// International Journal of Advanced Engineering Sciences and Technologies, 2011, vol. 11 (1), pp. 15-30.

7. Обзор NoSQL решений управления данными./ Мухина Ю. Р. // Управление в современных системах. 2013. No 1. С. 68-73.

8. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. / Ronald Weiss // - ORACLE WHITE PAPER, June 2012

9. IBM Documentation https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.db2.luw.licensing.doc/doc/c0056030.html (дата звернення 17.03.2017)

10. "Understanding the Cloud Computing Stack: SaaS, PaaS, IaaS," Rackspace, October 22, 2013.

11. "The NIST Definition of Cloud Computing" . National Institute of Science and Technology. Retrieved 24 July 2011.

12. PaaS Primer: What is platform as a service and why does it matter? / Brandon Butler // - Network World, February 11, 2013.

13. Cloud Service Based On Database Management System , Int. Journal of Engineering Research and Applications/ Monica Kadam,Pooja Jidge, Shubhangi Tambe,Pune,Ekta Tayade // ISSN : 2248-9622, Vol. 4, Issue 1(Version 3), January 2014, pp.303-306

14. Oracle Infrastructure and Platform Cloud Services Security //ORACLE WHITE PAPER, NOVEMBER 2016

15. ACHIEVING HIPAA COMPLIANCE WITH POSTGRES PLUS CLOUD DATABASE// EnterpriseDB Corporation, 2015.

16. URL:https://www.caspio.com/ (дата звернення 17.03.2017)

17. URL:http://w2.cleardb.net/(дата звернення 17.03.2017)

18. URL: https://mariadb.com/about-us/newsroom/press-releases/skysql-makes-highly-available-databases-easy-mariadb-enterprise (дата звернення 17.03.2017)

19. Год облачных СУБД/ Андрей Николаенко // Открытые системы.СУБД. –2013. – No 9. – С. 42–47.

20. URL: https://en.wikipedia.org/wiki/Shared_disk_architecture (дата звернення 17.03.2017)

21. URL: https://en.wikipedia.org/wiki/Shared_nothing_architecture (дата звернення 17.03.2017)

22. Parallel Execution with Oracle Database 12c Fundamentals // ORACLE WHITE PAPER, DECEMBER 2014

23. High Performance Parallel Database Processing and Grid Databases. Ch. 10/ David Taniar, Clement H. C. Leung, Wenny Rahayu, Sushant Goel //Wiley Publishing, 2008, isbn: 9780470107621

24. URL: https://dev.mysql.com/doc/refman/5.7/en/mysql-cluster.html

25. ElasTraS: An Elastic, Scalable, and Self Managing Transactional Database for the Cloud. /Sudipto Das,Shashank Agarwal,Divyakant Agrawal,Amr El Abbadi// UCSB Computer Science Technical Report 2010-04

26. The Teradata Scalability Story, A Teradata White Paper, EB-3031 0701, 2001, NCR Corporation

27. GRIDSCALE ® DATABASE VIRTUALIZATION SOFTWARE// TECHNICAL WHITEPAPER,2008 xkoto, Inc. Item: GS-WP-EN-20080930

28. Mark Bauer. Oracle8i Parallel Server Concepts, Release 2 (8.1.6) Part No. A76968-01, December 1999

29. Shared-Nothing vs. Shared-Disk Cloud Database Architecture. // Sunguk Lee // International Journal of Energy, Information and Communications Vol. 2, Issue 4, November, 2011

30. ORACLE ACTIVE DATA GUARD, REAL-TIME DATA PROTECTION AND AVAILABILITY// ORACLE WHITE PAPER, October 2015

31. Oracle GoldenGate 12c: Real-Time Access to Real-Time Information. //ORACLE WHITE PAPER, March 2015

32. Top Five Reasons to Choose SharePlex® Over Oracle GoldenGate/ Tom Chu. //Quest Software, November, 2011

33.Percona XtraDB Cluster Release5.7.17-29.20 Operations Manual (доступний на https://learn.percona.com/download-percona-xtradb-cluster-5-7-manual, дата звернення 17.03.2017)

34. Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987): Concurrency Control and Recovery in Database Systems, Addison Wesley Publishing Company, ISBN 0-201-10715-5

35. Open Group Standard DRDA, Version 5, Volume 3: Distributed Data Management (DDM) Architecture // ISBN: 1-931624-93-3 Document Number: C114

36. Consensus on Transaction Commit. Jim Gray and Leslie Lamport. Microsoft Research. 1 January 2004 revised 19 April 2004, 8 September 2005.

37. Maat: Effective and scalable coordination of distributed transactions in the cloud./ H. A. Mahmoud, V. Arora, F. Nawab, D. Agrawal, A. El Abbadi// PVLDB, 7(5):329–340, 2014.

38. Increasing the Resilience of Distributed and Replicated Database Systems./Keidar, Idit; Danny Dolev (December 1998) //Journal of Computer and System Sciences (JCSS) 57 (3): 309–324. doi:10.1006/jcss.1998.1566

39. An Improved Two-phase Commit Protocol Adapted to the Distributed Real-time Transactions / Xiai YAN , Jinmin YANG , Qiang FAN // PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review), ISSN 0033-2097, R. 88 NR 5b/2012.

40. Oracle Real Application Clusters (RAC)/Markus Michalewicz, Burt Clouse, John McHugh // Oracle White Paper. June 2013

41. Кузнецов, Сергей Транзакционные параллельные СУБД: новая волна (рус.) // Труды Института системного программирования РАН. — : Институт системного программирования РАН, 2011. — Т.20. — С.189—251.

42. АЛГОРИТМ ДИНАМИЧЕСКОГО МАСШТАБИРОВАНИЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ В ОБЛАЧНЫХ СРЕДАХ / М. Бойченко А.В., Рогожин Д. К., Корнеев Д. Г.// Экономика, Статистика и Информатика No 6 (2), 2014 с. 461-465

43. АВТОМАТИЗАЦИЯ МАСШТАБИРОВАНИЯ ВЫСОКОНАГРУЖЕННЫХ БАЗ ДАННЫХ MYSQL/Чистов В.А., Лукьянченко А.В // СОВРЕМЕННЫЕ НАУКОЕМКИЕ ТЕХНОЛОГИИ, : 6-2 2016 С. 315-319

44. Горобец В.В. «МАТЕМАТИЧЕСКИЕ МОДЕЛИ И АЛГОРИТМЫ ОПТИМИЗАЦИИ РАЗМЕЩЕНИЯ ДАННЫХ ТРАНЗАКЦИОННЫХ

СИСТЕМ» Диссертация на соискание ученой степени кандидата технических наук. Новочеркасск, 2015

45. ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ БАЗЫ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ КОНСИСТЕНТНОГО ХЕШИРОВАНИЯ/ Зернов А.С., Ожиганов А.А.// Известия высших учебных заведений. Приборостроение. 2017. Т. 60. № 3. С. 234-238.

46. SCALING OF LOAD WEB-APPLICATIONS/ Yakymets R.V., Yaremenko K.N. // Міжнародний науковий журнал. 2016. Т. 2. № 6. С. 45 – 47.

47. Кластерные СУБД [Электронный ресурс]/ Павлов Д.// Информационный бюллетень компании «Инфосистемы Джет» : [сайт]. URL: http://www.jetinfo.ru/stati/ klasternye#gl_2 (дата звернення: 17.03.2017).

48. БАЗЫ ДАННЫХ: СОВРЕМЕННЫЙ ПЕЙЗАЖ В ИСТОРИЧЕСКОЙ ПЕРСПЕКТИВЕ /Новиков Б.А., Графеева Н.Г., Михайлова Е.Г. // Компьютерные инструменты в образовании. 2016. № 1. С. 5-16.

49. ОБЗОР СОСТОЯНИЙ, ПРОБЛЕМ И ПЕРСПЕКТИВ ХРАНЕНИЯ И АНАЛИЗА ДАННЫХ В «ОБЛАКЕ»/Коваленко О.С.//Информатика, вычислительная техника и инженерное образование. 2011. № 5 (7). С. 39 – 49.

50. Построение и Адаптация NewSQL СУБД в частном «облаке»/А. С. Соболь // Вестник СибГАУ. No 4(50). 2013

51. Sharding by Hash Partitioning - A Database Scalability Pattern to Achieve Evenly Sharded Database Clusters /Caio H Costa, João Vianney, Paulo Maia, Francisco Carlos M. B. Oliveira//17th International Conference on Enterprise Information Systems (ICEIS 2015), At Barcelona, Spain DOI: 10.5220/0005376203130320

52. Oracle Partitioning in Oracle Database 12c Release 2 Extreme Data Management and Performance for every System// ORACLE WHITE PAPER MARCH 2017

53. Sharding by Hash partitioning. A database scalability pattern to achieve evenly sharded database clusters./Costa, C.H., Filho, J., Maia, P.H.M., Oliveira, F.//International conference on enterprise information systems (ISEIS), 2015

54. Oracle ® Database VLDB and Partitioning Guide 12c Release 1 (12.1) //Oracle Documentation E41057-12, July 2016

55. Understanding Query Performance in Accumulo/ Scott M. Sawyer, B. David O'Gwynn, An Tran, Tamara Yu // 2013 IEEE High Performance Extreme Computing Conference (HPEC)

56. TriAD: A Distributed Shared-Nothing RDF Engine based on Asynchronous Message Passing /Sairam Gurajada, Stephan Seufert, Iris Miliaraki, Martin Theobald // SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

57. Oracle® Database VLDB and Partitioning Guide 11g Release 2 (11.2)// Oracle Documentation E25523-01 September 2011.

58. URL: https://en.wikipedia.org/wiki/Online_transaction_processing (дата звернення: 17.03.2017).

59. URL: https://db-engines.com/en/ranking (дата звернення: 17.03.2017).

60. Исследование области применения неблокирующего алгоритма фиксации распределённых транзакций / Гусев Е.И. // – Вісник НТУУ "КПІ".Сер. Інформатика, управління та обчислювальна техніка. – 2012. Випуск 57. – С.76-80

61. Математическое моделирование распределённой кластерной системы использующей shared everything подход (Oracle RAC)./ Гусев Е.И. // - Вісник НТУУ "КПІ". Сер. Інформатика, управління та обчислювальна техніка. - 2014. Випуск 60. - с.106 – 113

62. Principles of transaction-oriented database recovery./Theo Haerder, Andreas Reuter // ACM Computing Surveys, Volume 15, Issue 4, December 1983, pp. 287 – 317

63. Towards Robust Distributed Systems. / Eric Brewer // Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 2000, p. 7

64. Brewer's conjecture and the feasibility of consistent,available, partition-tolerant web services.//Seth Gilbert, Nancy Lynch.//ACM SIGACT News, Volume 33 Issue 2, June 2002, pp. 51-59,

65. Errors in Database Systems, Eventual Consistency, and the CAP Theorem / Michael Stonebraker // BLOG@CACM, April 5, 2010,

66. Clarifications on the CAP Theorem and Data-Related Errors. / Michael Stonebraker // VoltDB.com, October 21, 2010, http://voltdb.com/blog/clarifications-cap-theorem-and-data-related-errors.

67. Weak Consistency and CAP Implications /Grigorik, Ilya // Igvita (24 June 2010)

68. TPC BENCHMARK C Standard Specification Revision 5.11// www.tpc.org , © 2010 Transaction Processing Performance Council (TPC) , February 2010.

69. BenchPress: Dynamic Workload Control in the OLTP-Bench Testbed / D. Van Aken, D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux// 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 1069-1073.

70. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases/ D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux // PVLDB, vol. 7, iss. 4, pp. 277-288, 2013

71. "Benchmarking OLTP/Web Databases in the Cloud: The OLTP-bench Framework/ C. A. Curino, D. E. Difallah, A. Pavlo, and P. Cudre-Mauroux,// Fourth International Workshop on Cloud Data Management, 2012, pp. 17-20.

72. Benchmark Overview OLTP-2 Version: 1.2 // White Paper. Fujitsu Technology Solutions. 2015

73. The Entity-Relationship Model: Toward a Unified View of Data/Chen, P.P. //ACM Trans. on Database Systems, Vol.1, No.1, March 1976, pp. 1-36.

74. The Relational Model of Data for Large Shared Data Banks/Codd, E. F. // Comm. of the ACM, Vol. 13 (6), 1970, pp. 377-387.

75. Моделирование трафиков и оценка скорости распределённого доступа в системах облачных вычислений с общим ресурсом на примере Oracle RAC / Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2015. – Випуск 62. – С. 32 – 42.

76. Naming and Synchronization in a Decentralized Computer System/ Reed, David P. // Massachusetts Institute of Technology, September 21, 1978.

77. URL:https://www.slideshare.net/MarkusMichalewicz/oracle-rac-internals-the-cache-fusion-edition (дата звернення: 17.03.2017).

78. URL: https://en.wikipedia.org/wiki/Decision_support_system(дата звернення: 17.03.2017).

79. Oracle® Database Concepts 11g Release 2 (11.2) // E16508-05 October 2010

80. URL: https://en.wikipedia.org/wiki/Shard_(database_architecture) (дата звернення: 17.03.2017).

81. 2Q: Low Overhead High Performance Management Replacement Algorithm / Theodore Johnson, Dennis Shasha // VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile. Morgan Kaufmann, 1994. pp. 439-450

82. RAC Performance Tuning./ Brian Peasland. //Rampant TechPress, January 25, 2015

83. Oracle8i Internal Services for Waits, Latches, Locks, and Memory. / Steve Adams // O'Reily, October 1999, ISBN-13: 978-1565925984

84. Oracle ® Database Concepts 12c Release 1 (12.1) E41396-14 September 2016 с. 3-24 – 3-25

85. Oracle® Database Administrator's Guide 11g Release 2 (11.2) E25494-01 September 2011 с. 23-3 – 23-6

86. URL: http://www.uml-diagrams.org/timing-diagrams.html (дата звернення: 17.03.2017)

87. URL: https://en.wikipedia.org/wiki/Stored_procedure (дата звернення: 17.03.2017)

88. Agile Database Techniques: Effective Strategies for the Agile Software Developer /Scott W. Ambler.// John Wiley & Sons 2012, с. 274 – 277

89. URL:https://en.wikipedia.org/wiki/Exception_handling (дата звернення: 17.03.2017).

90. Математическое моделирование распределённой кластерной системы использующей shared everything подход (Oracle RAC) / Е.И. Гусев // Вісник НТУУ «КПІ». Сер. Інформатика, управління та обчислювальна техніка. – 2014. – Вип. 60. – С.106 – 113.

91. Моделирование способов организации доступа к распределённым страницам памяти в системах облачных вычислений основанных на shared everything архитектуре / Е.И. Гусев // Вісник НТУУ «КПІ» .Сер. Інформатика, управління та обчислювальна техніка. – 2016. – Вип. 64. – С.133 – 137.